90

12

90

-: HAND WRITTEN NOTES:-

OF

1

ELECTRICAL ENGINEERING

-: SUBJECT:-

MICROPROCESSOR

MICROPROCESSOR

IES →

M.Tech → obj ≡ 5-6 Q (8085) + 1 Q (8086)

↳ Conventional ≡ 30 to 40 marks.

Definition :- * It is electronic device that fetch instruction from memory execute them & provide result.

* It is electronic device that have computing & decision making capability.

Memory :- ROM ≡ ROM

RAM ≡ Main Memory ≡ Memory.

Note →

→ A µp. can't perform any task on its own.

µP ≡ H/w µP + S/w.

ROM :- All system related information store in it

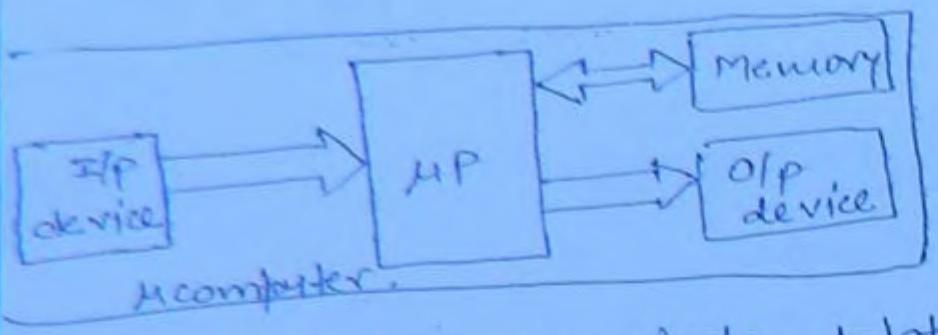
→ It come in the picture at the time of power switch on condition.

RAM :- 

Programs or instructions always feed in RAM. it is also called main memory or memory.

µ.computer → If all the task of CPU perform by µP then such type of device is known as µcomputer.

μcomputer ≡ μP + Memory + I/p device + O/p device.



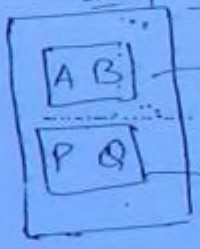
(4)

→ μ-computer on single platform is known as μ-controller (chip.).

ASIC ≡ Application specific integrated chip.
μ-controller is the example of ASIC design.

Machine Language :- If instructions ~~are~~ or command written in binary pattern, then such type of langue is called m/c language.

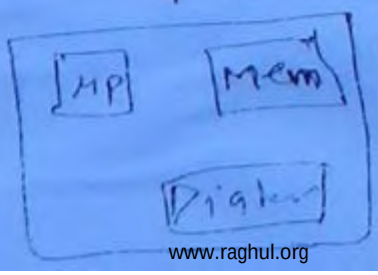
→ It is platform dependent language or m/c specific language.

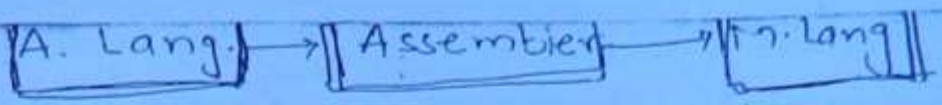


1010 ≡ samosa.
1011 ≡ Hit.
1010 ≡ walk.

Assembly Language :- If binary command replaced by english like word that is called "Mnemonic" such type of language is called Assembly langue.

→ It is also platform dependent langue.



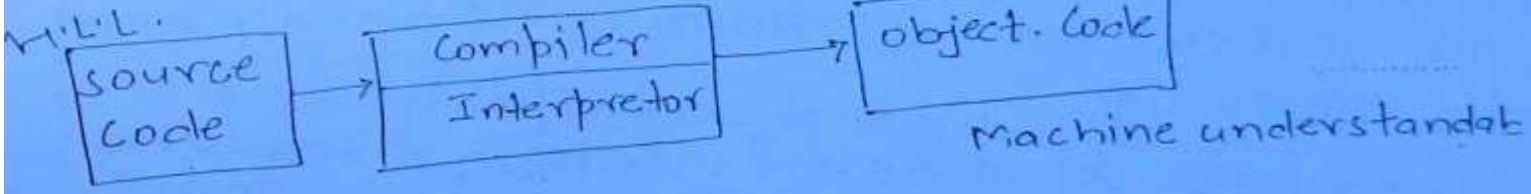


→ Assembler is a s/w that convert (translat
A.L → M.L.

→ If task is performed by manually then it is known as hand Assembly.

Low level language :-
All platform dependent language known as low leve language.
Ex→ All m/c language & assembly language.

High level language :-
All platform independent language known as hig level language.
Ex→ C, C++.



Compiler :- It read whole program at once prod it's object code, that executed by processor.
→ It is a s/w.
Ex→ C, C++.

Interpretor :- It reads one instruction at a tin produce it's object code, that is execute by processor, before reading next instru from source code.
Ex: BASIC. → It is a s/w.

Note -

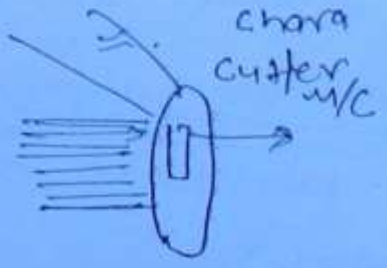
→ 8085 is commonly known as 8085 µp.

→ It is based upon NMOS tech. (6)

→ It is improved version of 8080.

→ It is 8 bit µP.

Bit of µP.



→ Data executed by µP in one w/c cycle is bit of µP

→ Size of A.L.U. (Arithmatic & logic unit) is also known as bit of µp.

BUS → It is group of (parallel combination) metal wire that is used for interfacing b/w two different device.

→ All instruction of 8080 µp is exactly comm in 8085 µP. → upward compatibility

Instruction set of 8080 + RIM + SIM = Inst. set of 8085

µP	bit of µP.	Used tech
4004	4 bit	PMOS
8008	8 bit	NMOS
8080	8 bit	NMOS
* 8085	8 bit	NMOS.
8086	16 bit	HMOS (H= High density Comp
8088	8/16 bit	HMOS.

8088 is the externally 8 bit & internally 16 bit µp

8086 ⟶ 16 bit

80286 ⟶ 16 bit

80386 ⟶ 32 bit

80486 ⟶ 32 bit.

80586 ⟶ 64 bit μP ≡ Pentium.

(7)

(i) Pentium.
(ii) Pentium Pro
(iii) Pentium II.
(iv) Pentium III.
(v) Pentium IV.

} 64 bit μP.

Note→ MOV BC ≡ 4T.

$1T = \frac{1}{f}$.

exe. time = $4T = \frac{4}{f}$.

speed $\propto \frac{1}{\text{e.time}} \propto \frac{1}{\frac{4}{f}}$.

speed $\propto f$.

$\propto$ bit of μP.

Architecture of 8085

(28)

→ External Arch. → (Pin diagram)
→ Internal Arch.

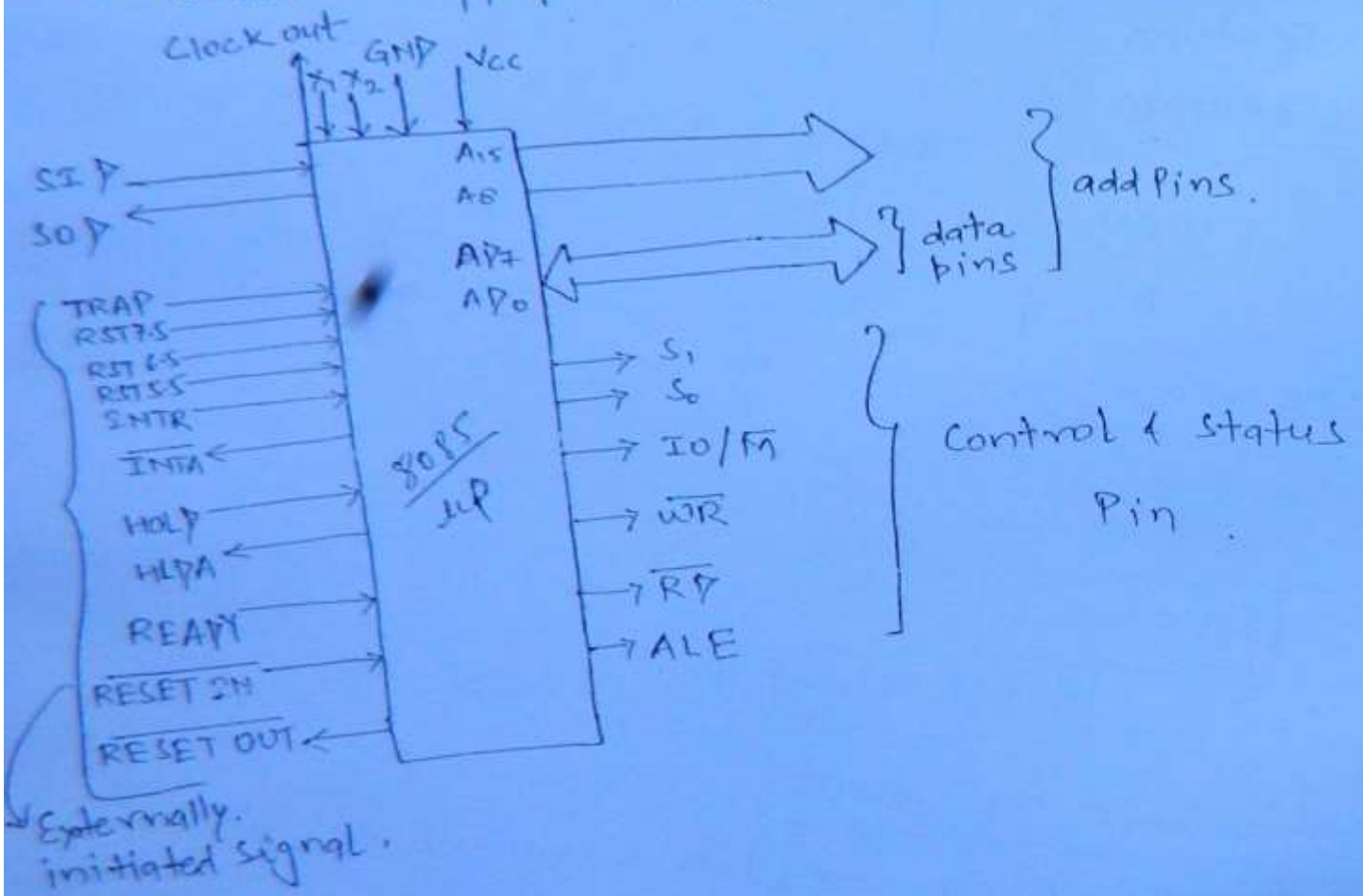
External Arch.→

→ Pin diagram.
→ It is 40 pin I.C.
→ Pin no. is 20 is GND pin.
→ There is no chip enable pin in 8085

→ All 40 pins are paper there 7 divide in six groups.

(8)

* Add Pins.
* Data Pins.
* Control & status Pin.
* Serial data transfer Pin.
* Externally initiated signal Pin.
* Power supply & freq. Pin.



Note →

① No. of pins active low.

IO/M, WR, RD, INTA, RESET IN.

* Advantage of active low

→ Power consumption is less

→ Effect of external noise is less

→ Interfacing will be better.

(2) No. of Pins direction outward ≡ 27.

(3) No. of Pins direction inward ≡ 21.

(4) Address Pins. (9)

→ 16
→ Unidirectional & outward.
→ max^m no. of memory location that can be interfaced = 2^{16} location.

(5) Data Pins.

→ 8
→ Bidirectional.

→ Lower 8 pins of address pins is can also be used as data pins.

→ Max. memory that can be interface with 8085 ≡ $2^{16} \times 8$ bits.
≡ 2^{16} Byte. = 65536 Byte.

2^{10} = kilo. ≡ 2^6 K. Byte.
2^{20} = mega. ≡ 64 K. Byte.
2^{30} = Giga.

memory word size → max^m no. of bits that can be stored at particular memory location that is known as word size.

Q: $2^{16} \times 8$ bits memory. Find max^m no. of Have wave Pins. RAM.
= 16 + 8 + V_{cc} + V_{GND} + WR + RD

add. { n : [$2^n \times d$] : d } data. (20 or 29) may or may not be

→

(5) Control & Status sig pins :-

→ Control & status pins direction mixed direction.

* S1 & S0 } By measuring logic on that we can find out which m/c cycle going on inside the µP.

(10)

* IO/$\overline{M}$ = 1 = data transfer from IO.
= 0 = data transfer from memory.

* $\overline{WR}$ = 0 = write operation can be performed.
$\overline{RD}$ = 0 = Read operation can be performed.

Group of bit.

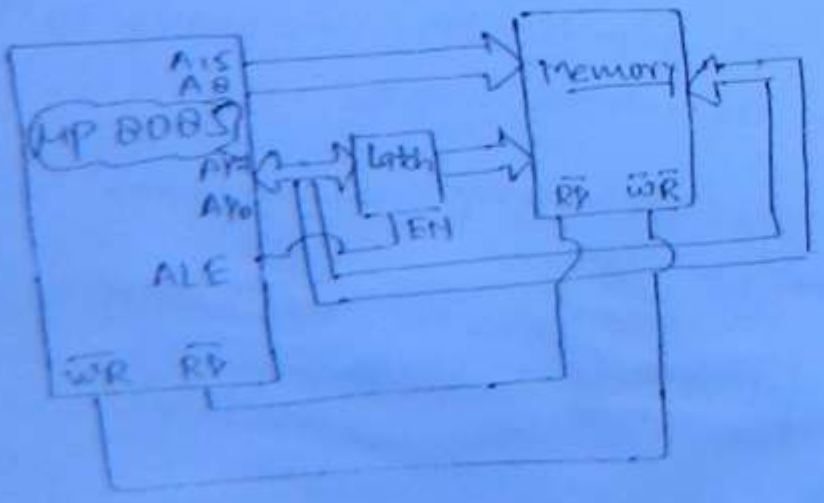
1111 = nibble.
11 = dite
1111 0111 = Byte

$\overline{WR}$ = 0
$\overline{RD}$ = 0 } X.

$\overline{WR}$ = 1
$\overline{RD}$ = 1 } ✓.

(6) * ALE (address latch Enable) :→

ALE = 1 $AD_7 - AD_0$ = addl.
= 0 $AD_7 - AD_0$ = data.



Note:→ A sequential ckt maybe a latch if it is level trigger or follows its characteristic eqn in definite duration. (11)

→ A seq. ckt may be a flip flop, if it is edge trigg. or follows its char. eqn in definite time instant.

* (ALE = 1)
  AD7 – AD0 ≡ add.

  (ALE = 0)
  AD7 – AD0 = data.

* By the use of ALE lower 8 pins of add. pins multiplexed/ demultiplexed as data pins.

* multiplexing & Demultiplexing = Time division multipl
  = T.D.M.

* Serial data transfer pin→

  SID ≡ serial i/p data through this pin by use RIM inst
  SOD ≡ serial o/p data through this pin. by use SIM inst

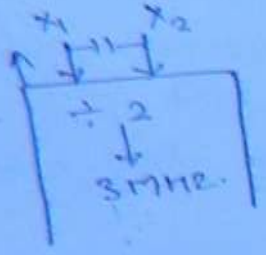
  RIM ≡ Read interrupt mask.
  SIM ≡ Set interrupt mask.

* Power supply & freq. pins→

  V_{CC} = +5V.
  V_{GD} = Ground Pin.
  X ≡ operating freq = 3MHz.

→ To get 3MHz clock freq a 6MHz crystal

oscillator connected b/w X_1 & X_2 & use a internally divide by two ckt to get 3MHz. clock freq.



* Clock out → By use of this pin clock freq. provide by µp. to other interfaced peripheral for better synchronization.

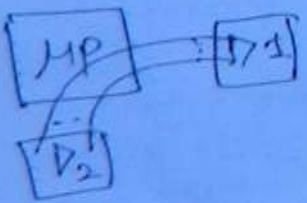
* Externally initiated signal Pins

5, 6, 7, 8.

Interrupts
- TRAP
- RST 7.5
- RST 6.5
- RST 5.5
- INTR

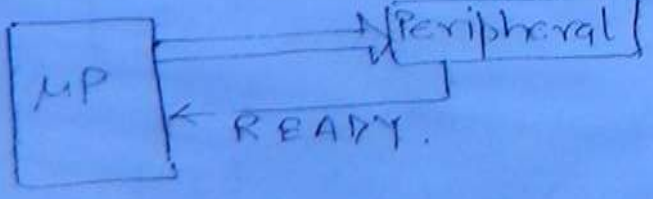
5 pin or 6 pin (Reset)
↓
H/w interrupt.
* s/w interrupt.

HOLD → By use of this pin DMA (direct memory access) like peripheral make request to µp for relinquish to its data pins.



READY

slower.



interface with 8085 µP.

RESET IN:- By use of this pin Reset command gives to µP for reset.

Note:- As soon as 8085 µP receive reset comman it generate reset out command for reset of other & interfaced peripheral.

INTERNAL ARCHITECTURE.

* ALU Arithmatic & logic unit.
  - → Accumulator.
  - → Timing & control ckt.
  - → Registers.
  - → Register array.
* Interrupt control register.
* Temporary Registers.
* Latches.

Registers.

- General Purpose Register (user Accessable).
  - → B
  - → C
  - → D
  - → E
  - → H
  - → L

  } 8bit.
- special Purpose Regist
  - (user accessable)
    - 8bit → Accumulator (A)
    - 8bit → Flag Reg - or Status Reg (F)
    - 16bit → Stack Pointer (SP)
    - 16bit → Program Counter (PC).
  - (user not accessable)
    - 8bit → Temporary Reg.
    - → Interrupt Control Reg.
    - → Latches etc

user accessable = Use of in programming.

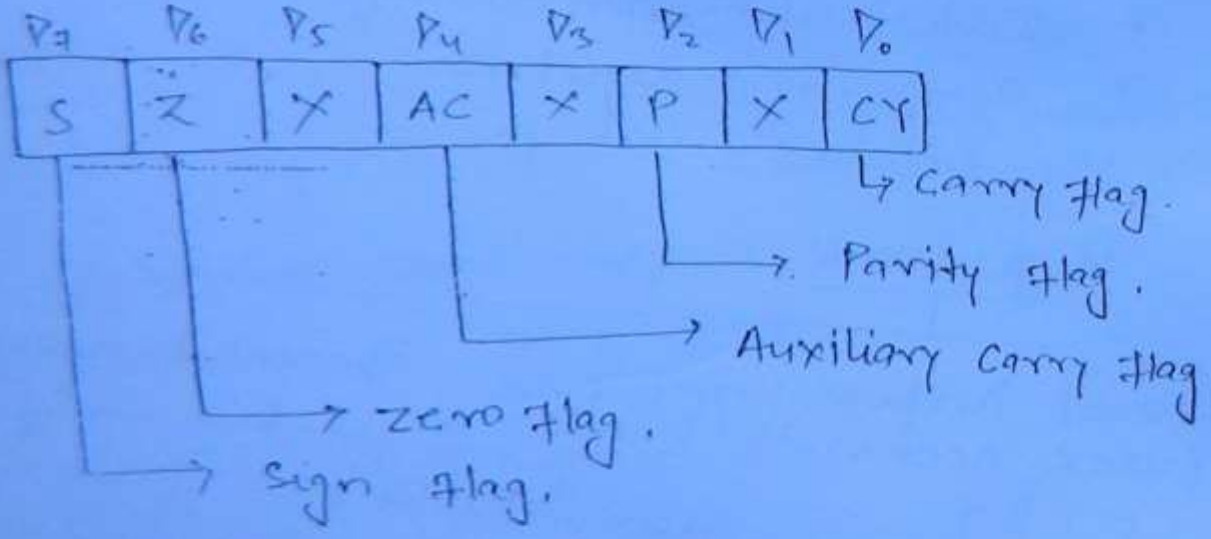
Note→ In 8085, 8, 8 bit register & 2, 16 bit register are used, that are user accessable.

(i) Accumulator→

* It is 8 bit special type of Reg.
* In A & L (Arithmatic & logic) operation b/w the two no., then one no. always from accumulator & result will also store in Accumulator.

(ii) Status or flag Reg→

* It is 8 bit special type of Reg.
* It's definite bits also works as flags.
* In 8085, 5 flags are define at definite bit of flag Reg.



x = don't care.

Note → status of flag affect according to the condition generate in Arithmatic & logic operation.

(a) Sign flag (S)

If in the final result of arithmatic & logic operation, D7 bit is '1', then sign flag is set i.e. no. is -ve. (15)

S = 1.

→ If D7 bit is or MSB = 0, then S = 0 or no. is +ve

(b) Zero flag (Z).

If in the final result of ALU operation all bits are zero, then Z = set (1).

otherwise Z = Reset (0).

	D7	D6	D5	D4	D3	D2	D1	D0
	d7	d6	d5	d4	d3	d2	d1	d0
S = 0	0	0	0	0	0	0	0	0
	Z = 1.							
S = 1	1	0	0	0	0	0	0	0
S = 1	1	1	1	1	1	1	1	1

} → Z = 0.

→ There is no relation among the flags, they affect according the result of ALU operat respectively.

(c) Parity flag (P) If in the final result of ALU operation, total no. of 1's is even, then parity flag will get set - P=1 >s

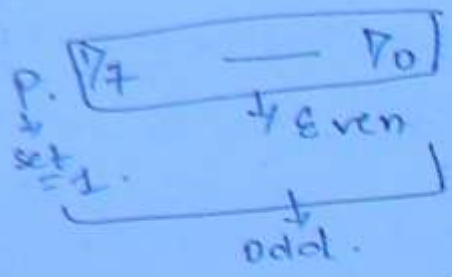
if total no. of 1's is odd, P = 0 - Reset.

10 10 0011 P = 1

10 11 0011 P = 0

0 0 0 0 0 0 0 0 P = 1

→ 8085 µp. based upon odd parity system.



(16)

② Carry flag (CY) If in the ALU operation carry generated / discarded from D7 bit or from MSB then, CY = set (1).

otherwise, CY = 0 (Reset).

D7	D6	D5	D4	D3	D2	D1	D0
d7	d6	d5	d4	d3	d2	d1	d0

1 (1, 1) | × (0, 0) → 1

Note→ In substraction operation carry flag works as borrow flag.

→ If in the substraction operⁿ, borrow is taken at MSB or D7 bit, then borrow flag will get set; CY = 1.

otherwise, CY = 0.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	0	1	0	
0	0	1	0	0	0	1	
1	1	1	1	0	0	1	

CY = 1.

(c) Auxiliary carry (AC) flag

During the ALU operation if carry passes from $D_3 - D_4$ or lower nibble to upper nibble then AC flag will set.

AC = 1.

otherwise AC = 0.

msB								LSB
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	

Note → ⊛ five flag divide in two category.

(i) status of flag that affect according final result of ALU operation - (S Z P)

(ii) status of flag that affect according condtn generate during the operation. (CY AC)

⊛ Out of five flag, status of 4 flag can be used by programmer during the programming (S, Z, P, CY)

→ status of 'AC' flag not available for program

→ status of 'AC' flag used internally for the BCD adjustment of content of accumulate at the time of execution of DAA instructio

DAA → Decimal adjustment of content of accumulator.

BCD = Binary Coded decimal.

→ It is not binary no. it is binary code decimal.

→ Every digit of ~~bin~~ decimal code in 4 bit of binary.

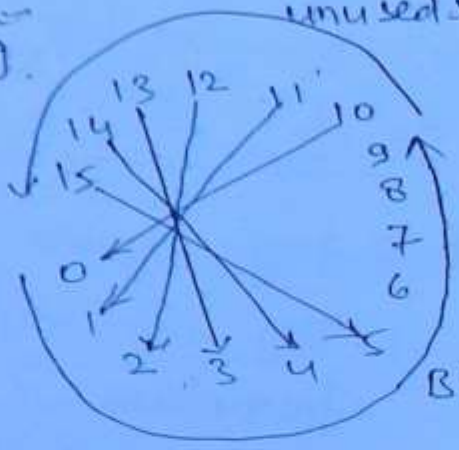
→

25	=	0010	0101
37	=	0011	0111
62		0101	1100 → move thang add 6.
			0110
		0110	0010
		6	2

(18)

0 1 2 3 4 5 6 7 8 9.

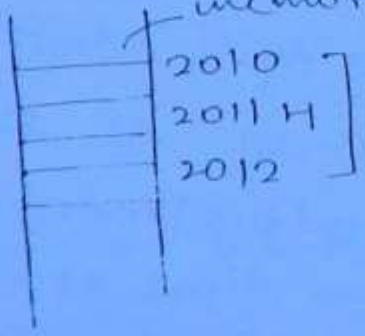
9 → 10
9 + 1 = 10.



(iii) Stack Pointer (SP).

* It is 16 bit special type of reg.
* S.P. always hold the address of top of stack that define in main memory.

2010 [0010 0000 0001 0000
2011 [0010 0000 0001 0001



Note→ S.P. stores data, & that data is the add. of tope of stack.

(iv) Program Counter (PC)

* It is 16 bit special type of Reg.
* P.C. will always hold the address of next execati Instruction to be fetch.

Ex: MOV BC
MVI D 20H
LXI H 10 69H.

Note → * If a 16 bit no. feed in memory then lower order byte at lower order add. & higher order byte always at higher order add.

* If a 16 bit no. store in register pair then 8 bit reg. coupled in specific manner.

BC } ≡ B.
DE } ≡ D.
HL } ≡ H.

* Higher order byte of 16 bit no. will always store in higher order register. B } D } H }

& lower order byte always store in lower order register. C } E } L }.

* (PSW) = Program status word.
→ It is 16 bit user define register.

≡ A F → Flag Register
↓
Accumulator

* PROGRAMMING.

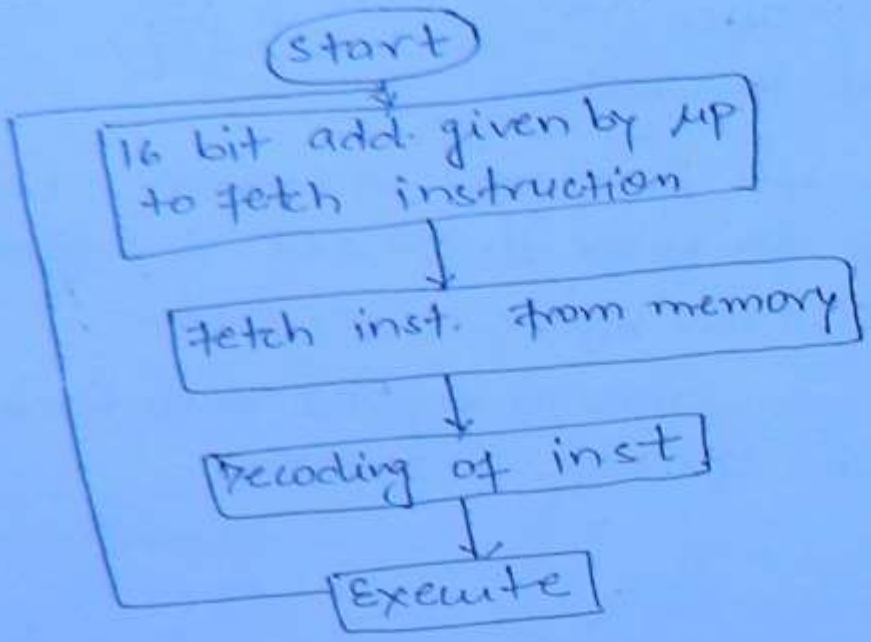
Instruction → It is command that gives to up to perform a specific task on specific data

Format of instruction →

Table : opcode operand ; discription

(operational code)

Flow Chart of Execution →



IWS ≡ Instruction word size.

Total memory location required to feed instruction in memory is called IWS.

→ On the basis of IWS inst. are of three type.

(i) 1 Byte inst.
(ii) 2 Byte inst.
(iii) 3 Byte inst.

Condition.

① If in the instruction Register, Register pair or no-operand.

IWS ≡ 1 Byte.

MOV BC
LXAX B
NOP
} 1 Byte.

② If in the instruction 8 bit no., either as an address or as data is given then

IWS ≡ 2 Byte.

MVI B 26H
IN 56H
} IWS = 2 byte.

(3) If in the instruction 16 bit no. either a add. or as data is given then.

IWS = 3 byte.

LXI B 2016H } IWS = 3 byte.
LDA 5926

Instruction →

- NAME.
- FORMAT.
- IWS.
- OPERATION.
- STATUS OF FLAG.
- ADDRESSING MODE.
- MACHINE CYCLE.
- T-state.

MOV BC
mov DE
} diff. inst.

opcode

Note:- In 8085 74 opcodes (operational code) are available, by that 256 inst. can be def But only 246 inst. are available in 8085.

74 opcodes has 11 Groups →

1st Group

(1) 8 bit data transfer inst.

(a) MOV Rd, Rs.

Rd = destination Reg } A, B, C, D, E, H, L & M.
Rs = source Reg.

→ MOV BC
MOV DE

→ IWS = 1 Byte.

→ operation: Within this inst will exec

then content of source Reg. (Rs) will pass in Rd Reg. $[R_d] \leftarrow [R_s]$.

Ex-1 [A] = 26H, [C] = F6H

MOV A, C

[A] = F6H

[C] = F6H.

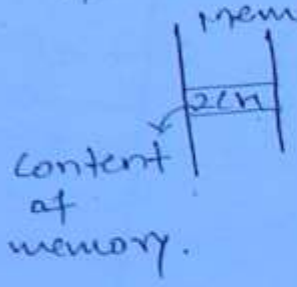
→ M :- It is 8 bit user defined register in main memory. & its address is the content of HL register pair.

[H] = 16H

[L] = 9DH

→ MOV B M.

[B] = 2CH.



⑧ MVI R, 8 bit data

→ R ≡ A, B, C, D, E, H, L & M.

→ operation :- When this instruction will execute 8 bit data given in the instruction will store in R.

[R] ← 8 bit data.

Ex-1 MVI D, 69H.

[D] = 69H.

→ 1WS ≡ 2 Byte.

* Note:- If opc has last word is 1, then no. given in data. inst. is always data, otherwise no. is address.

→ LXI B 1015H
→ 3M 52H
↓ data
↓ add.

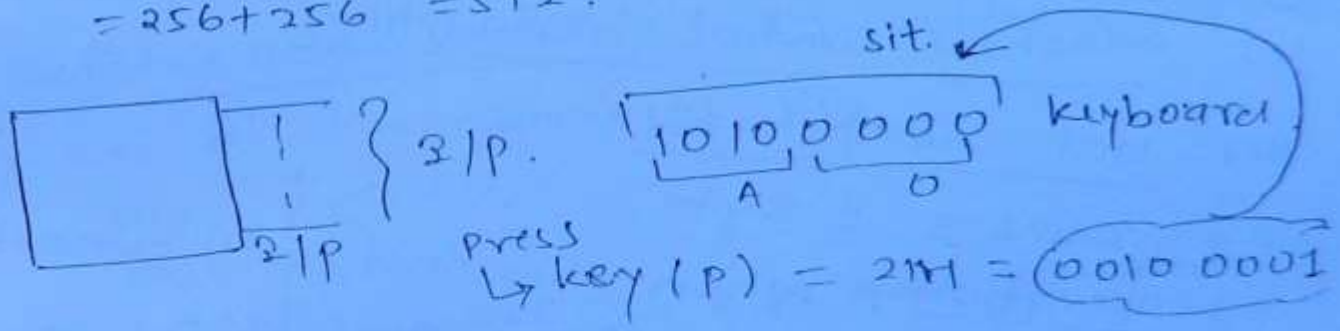
(c) IN 8 bit port add.

→ IWS ≡ 2 Byte.

→ operation: When this instruction will execut then 8 bit data available at the port ad given in instruction store/pass in Accumula

Note → In 8085 i/p & o/p ~~code~~ port add. is of 8 bit so max^m no. of i/p devices that can be connect = 25 & " " " o/p " " " " = 2^8 = 25

→ max. no. of i/p + o/p devices that can be connecte = 256 + 256 = 512.



IN A0H

→ [A] = 21H.

(d) OUT 8 bit port add

→ IWS ≡ 2 Byte

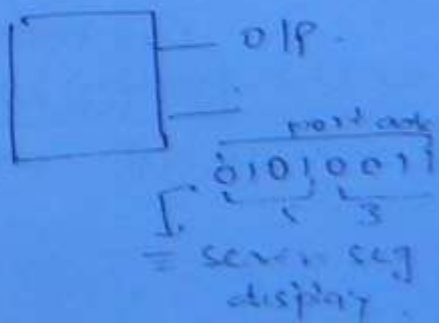
→ operation: When this inst. will execute content of accumulator will available a 8 bit port add. given in the instructio

Ex: [A] = F2H (Accumulator).

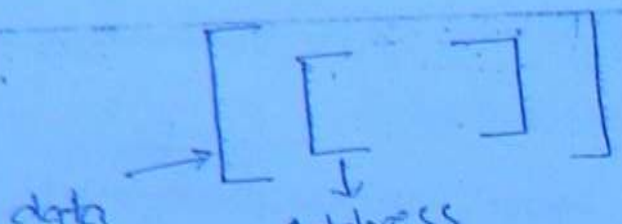
OUT 53H

53 port add: F2.

1	1	1	1	0	0	1	0



Note



Ex: [B]

data of B.

(24)

[[53]] ≡ F2.

data at the address of 53

→ Note All above instruction are data transfer instruction, so status of flag will not affect.

2nd Group:

(i) Machine control instruction

(a) NOP no operand →

→ IWS ≡ 1 Byte.

→ operation: When this inst. will execute µp. will not perform any task.

→ It is used for creation of delay.

NOP ≡ 4T = $\frac{4}{f}$ sec

(b) HLT no operand:

→ IWS ≡ 1 Byte

→ operation: When this inst. will execute further increment of P.C. will stop. it means after the execution of HLT inst., execution of program will terminate.

2000H MOV BC
2001 MOV D,E
2002 MVI A 29H
2004 HLT
2005 —— PC=2005

PC ← PC+1

Note → All above inst. is machine control inst. so status of flag will not affect

Addressing Mode

Form of address of data given in the inst is known as addressing mode.

(i) Register add. mode → If address of data given in the form of Reg.

Ex MOV BC
MOV D,E

(ii) Direct add. mode → If add. of data directly given in instruction.

Ex IN 25H.

(iii) Immediate Add. mode → If data itself given in the inst.

Ex MVI B 20H.

(iv) Note → If op. code has last chara. 'I' the it is immediate addressing mode & vice-versa is not true.

(iv) Register Indirect / Indirect Reg / Indirect add. mode

If add. of data given in the form of content of Register.

Ex → MOV B M.

(v) Implicit addressing mode / Implied add. mode

If add. of data is not required & it is defined in opcode.

Ex NOP, HLT, CMA.

3rd group.

(i) 8 bit Arithmatic instruction

(a) ADD R.

→ Ins ≡ 1 Byte.

→ R = A, B, C, D, E, H, L, & M.

→ operation: When this inst. will execute content of 'R' will get added [A] & final result will store in [A].

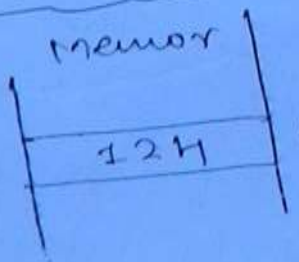
[A] ← [A] + [R].

Ex [A] = 29H.
[B] = 56H.
ADD B.

[A] = 0010 1001
[B] = 0101 0110
[A] = 0111 1111 = 7FH

R = [B] = 56H

Ex [H] = 20H
[L] = F6H
[A] = 10H.
ADD M



[A] = 00010000
[M] = 00010010

* R ≠ M = Register add Mode.
R = M = Indirect " "

ADD M.

(b) ADI 8 bit data

→ Bws ≡ 2 Byte.

→ operation :- When this inst. will execute, 8bit data will 'add' in content of acc. & result will store in [A].

[A] ← [A] + 8 bit data

→ Add mode ≡ Immediate.

(c) SUB R

→ Bws ≡ 1 Byte

→ R ≡ A, B, C, D, E, H, L & M.

→ operation :- [R] will get substracted from content of accumulator & result will store in Accu

[A] ← [A] - [R]

→ Add mode → R ≠ M Register add.
R = M Indirect Reg. add.

(d) SUI 8 bit data

→ Bws ≡ 2 Byte

→ operation :- 8 bit data given in the inst. will get substracted from content of [A] & result will store in [A].

[A] ← [A] - 8 bit data

→ Adal. mode → Immediate.

(e) INR R

→ Bws ≡ 1 Byte.

→ R ≡ A, B, C, D, E, H, L & M.

→ operation: Content of R increased by 1 result will store in R.

$[R] \leftarrow [R] + 1_{LSB}$.

→ Add. mode: R ≠ M → Register add. mode.
R = M → Indirect Reg. addl.

Ex: MVI B 29H
INR B.

→ [B] = 0010 1001
[B] = 0010 1001
1
0010 1010
2 A
[B] = 2AH

(7) DCR R.

→ Ins = 1 Byte.
→ R = A, B, C, D, E, H, L, & M.
→ operation: Content of R decreased by 1 & result will store in R.

$[R] \leftarrow [R] - 1_{LSB}$.

→ Add. mode: R ≠ M → Register add. mode.
R = M → Indirect add. mode.

Note: (1) ADD, ADI, SUB, SUI will affect status of all flag.
(2) INR & DCR will affect only status of four flags [S, Z, AC, P].

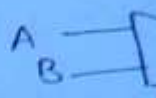
→ INR & DCR will not affect status of Carry flag.

[B] = FF B = 1111 1111
INR B. 0000 0000

S = 0
Z = 1
P = 1
AC = 1
CY = Previous

(i) 8 bit logical instruction.

(1) ANI operation:



$\Rightarrow Y = A \cdot B$.

A.B	Y
0 0	0
0 1	0
1 0	0
1 1	1

(a) ANA R

→ IWS = 1 Byte.
→ R = A, B, C, D, E, H, L & M.
→ operation: Content of R will get AND operation with content of [A], bit by bit & result will store in A.C.

Ex:
```
MVI A  56H   → [A] = 56H
MVI D  29H   → [D] = 29H
ANA D
```

```
[A] = 0 1 0 1 0 1 1 0
[D] = 0 0 1 0 1 0 0 1
-----------------------
[A] = 0 0 0 0 0 0 0 0
```

[A] = 00H.
[D] = 29H.

→ Add mode: R ≠ M → Reg. Add. mode.
R = M → Indirect Reg. add.

(b) ANI 8 bit data

→ IWS = 2 Byte.
→ operation: 8 bit data given in inst. will get AND operation with content of [A] bit by bit & result will store in [A].
→ Add. mode: Immediate add. mode.

(3) OR operation :

$\Rightarrow Y = A + B$.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

30

(a) ORA R

→ Ins ≡ 1 Byte.

→ R = A, B, C, D, E, H, L & M.

→ operation : [R] will get OR operation with conten of [A] bit by bit & result will store in [A].

→ Add. mode : R ≠ M → Register Add. Mode.
R = M → Indirect Reg. Mode.

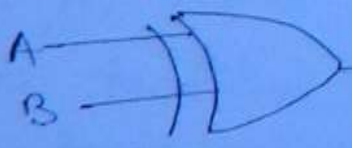
(b) ORI 8 bit data

→ Ins ≡ 2 Byte.

→ operation : 8 bit data will get OR operati with content of [A] bit by bit & result will store in [A].

→ Add. mode : Immediate Addressing.

(3) ExtOR operation :



$Y = A \oplus B = \bar{A}B + A\bar{B}$.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

(a) XRA R

→ 1WS ≡ 1 Byte

→ R ≡ A, B, C, D, E, H, L & M.

→ operation: Content of R will get EX-OR operation with [A], bit by bit & result will store in [A].

→ Add. mode: R ≠ M → Reg. add. mode.
R = M → Indirect add. Mode

Ex: [A] = 26H, [D] = D1H

XRA D.

[A] = 0 0 1 0 0 1 1 0
[D] = 1 1 0 1 0 0 0 1
[A] = 1 1 1 1 0 1 1 1
[A] = F7H.
[D] = D1H.

(b) XRI, 8 bit data

→ 1WS ≡ 2 Byte.

→ operation: 8 bit data in instruction will get EX-OR operation with [A], bit by bit & result will store in [A].

→ Add. mode: Immediate Add. mode.

(c) CMA no operand

→ complement of accumulator.

Note: $A \xrightarrow[1]{0} \triangleright \xrightarrow[0]{1} Y = \bar{A}$.

→ 1WS ≡ 1 Byte.

→ operation: content of [A] will get complemente bit by bit & result will store in [A]

→ Add. mode: Implicit.

Ex: [A] = 28H. | [A] = 00101000 (32)

CMA. [A] ← [$\bar{A}$] = 11010111 = D7H

Note :-

	S	Z	AC	P.	CY
ANA	✓	✓	1	✓	0
ANI	✓	✓	1	✓	0
ORA	✓	✓	0	✓	0
ORI	✓	✓	0	✓	0
XRA	✓	✓	0	✓	0
XRI	✓	✓	0	✓	0
CMA	X	X	X	X	X

✓ ≡ According to the result.

X ≡ Not affect

1 = set.

0 = Reset.

* ① CMA will not affect status of any flag.

② AND, OR, EX-OR will always reset the Carry flag.

③ AND operation always set the AC flag.

④ OR & EX-OR operation will always reset the AC flag.

⑤ AND, OR, EX-OR affect other flag as per result

Question :- FF00 MVI A 23H

FF02 MVI B 32H.

FF04 XRA B

FF05 ADI 00H

FF07 HLT

FF08

After the execution of HLT inst. value of PC = ?, B = ?, PSW = ?.

Sol: PC = FF08H.

→ [A] = 23H.

→ [B] = 32H

→ [A] = 0 0 1 0 0 0 1 1

[B] = 0 0 1 1 0 0 1 0

[A] = 0 0 1 0 0 0 0 1, [B] = 32H

[B] = 32H.

→ 88H = 1 0 0 0 1 0 0 0

[A] = 1 0 0 1 1 0 0 1

9 9

PSW = A F → Flag. Reg.
↓ Accumulator

= 9984H.

S	Z	X	AC	X	P	X	CY
0	0	0	1		0		0

1 0 0 0 0 1 0 0

8 4

X = 0

we have taken don't care.

Q:2 MVI A 2AH.

ADD A

ORI AF

INR A

CMA.

After the execution of progr. status of flags.

Soln → [A] = 2AH

[A] = 0 0 1 0 1 0 1 0

[A] = 0 0 1 0 1 0 1 0

[A] = 0 1 0 1 0 1 0 0

AFH = 1 0 1 0 1 1 1 1

[A] = 1 1 1 1 1 1 1 1

INR A = 1

[A] = 0 0 0 0 0 0 0 0

S	Z	AC	P	CY
0	0	1	0	0
1	0	0	1	0
0	1	1	1	Pre=0

Flags.

0 1 1 0

IES conv. 5 marks.

Q: Write one instruction that make content of ACC 00H regardless of its previous status.

Sol: (i) MVI A 00H.

(ii) SUB A.

(iii) ANI 00H.

(iv) XRA A → [A] = 10110100 ⊕ 10110100 = 00000000

Q: After the arithmatic operation b/w the two no., status of flag Reg = BBH. then after the arithmatic operation content of [A] maybe

S	Z	X	AC	X	P	X	CY
1	0	1	1	1	0	1	1

(A) 75 H
(B) 65 H
(C) DB H
(D) B6 H.

C → 1101 1011 → Parity → 1 (Even)
D → 1011 0110 → Parity → 0 (Odd)

(Parity)

M. 2nd.

* INSTRUCTION CYCLE, M/C CYCLE & T-State

(i) Instruction Cycle: Total time required for execution to complete execution of one instruction is known as instruction cycle.

→ Every inst. cycle is the combination of one or more than one m/c cycle.

m/c cycle: During the execution of instruction different type of task perform, is known as m/c cycle. (35)

→ In 8085 µP, six types of m/c cycle are defined.

(a) op-code fetch m/c cycle (F/S) :-
or
machine code fetch m/c cycle

Time required for execution to fetch op-cod (m/c code) regarding a instruction from memory is known as op-code fetch m/c cycle.

→ MOV BC → 1 Byte → m/c code.
CMA → 1 Byte → op-code.

Note → * It is first or only first m/c cycle of every instruction.
* op-code fetch m/c cycle, is the special case of memory read operation.

* F ≡ 4T
S ≡ 6T.

T-state. $1T = \frac{1}{\text{f oper. freq.}}$ sec

* m/c code regarding, opcode fetch m-cy that req. 6T state for m-code →

{ CALL, RET
RESTART
INX, DCX
SPHL, PCHL
PUSH

"CRISP"

(b) Memory Read M.Cy (R) :-

Total time required for execution to read 8 bit data from memory.

R = 3T.

(c) Memory write M.Cy (W) :-

Total time required for execution to store 8 bit data in memory.

W = 3T.

(d) Input read M.Cy.(I) :-

Total time required for execution to read 8 bit data from i/p port.

I = 3T

(e) o/p write M-Cy (O) :-

Total time required for execution to make available 8 bit data at o/p port.

O = 3T

(f) Bus idle M.Cy. (B) :-

Duration for which buses of µp will be in idle condition, during the execution of some specific inst.

B = 3T

Note :- This m/c cycle is required only in DAD instruction.

op-code Fetch m/c (F/S)	0	0	1	1	1
Memory Read M-Cy. (R).	0	0	1	1	0
Memory write M.Cy. (W).	0	1	0	0	1
Input read M-Cy (I).	1	0	1	1	0
o/p write M-Cy (O)	1	1	0	0	1

(37)

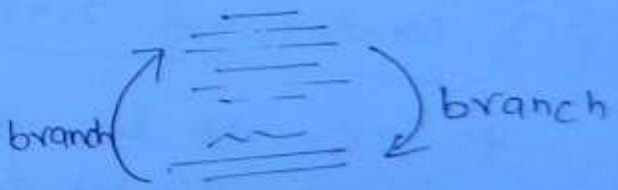
S_1	S_0	
1	1	op code fetch M. Cy.
1	0	Read operation.
0	1	write operation.

S_1 S_0 = 0 0 ≡ Bus idle M. Cycle.

Date 0/12/11

Branch operation ⇒

Loop is the special case of branch operation.



→ In 8085, there exist 3 instrs. defined for branch operation.

1. JMP
2. CALL
3. RESTART

JMP → It is of two types-

1. unconditional jmp inst
2. conditional jmp inst.

(i) unconditional JMP

* JMP 16 bit add.

→ IWS = 3 Byte.

→ operation: When inst. will execute, control/execution will JMP at 16 bit add (vector location) given in the instruction.

Ex. 1000H MOV BC
1001: MVI D, 27H
1003: JMP 2089H

1003 JMP H
1004 89H
1005 20H
1006

PC ← 2089

Imp

→ Add. Mode → Immediate Add. Mode.

→ It is data transfer instr., no status of flag will not affect.

(ii) conditional JMP instruction

* JC 16 bit add.; JMP inst. will execute if CY flag = 1
JNC 16 bit add.; " " " " CY = 0
JZ 16 bit add.; " " " " Z = 1
JNZ 16 bit add.; " " " " Z = 0
JP 16 bit add.; " " " " S = 0
JM 16 bit add.; " " " " S = 1
JPE 16 bit add.; " " " " P = 1
JPO 16 bit add.; " " " " P = 0

→ IWS = 3 Byte.

→ Operation; If condition satisfied JMP inst will execut otherwise skip it.

1005 JPE 16 bit add.

1005 JPE
1006 ——
1007 ——

→ Addl. mode → Immediate Addl. mode.

→ status of flag will not affect.

* Inst.	M.C.	T-state.
(i) MOV BC	F	4T
(ii) MVI D, 29H	F R	7T
(iii) MOV B M	F R	7T
(iv) MOV M, D	F W	7T
(v) NOP	F	4T.
(vi) IN 2FH	F R I	10T.

1000; (IN)
F. 1001 2F → Port addl.
(R) Controller → goes to this (F).
(FR2).

(vii) OUT 25H	F R O	10T.
3000 OUT.		
3001 25	F	4T.

(viii) ADD B

1000H ADD B
Fetch.

	Instr	M/C	T-State
(9)	ADD M	FR	7T.

(40)

2000 ADD M

fetch, then goes to add of HL (Read M) then add.

(10)	SUB C	F	4T
(11)	SUB M	FR	7T
(12)	INR B	F	4T
(13)	INR M	F. R. W.	10T.
(14)	DCR D	F	4T.
(15)	DCR M	F R W	10T.
(16)	SUI 29H	F R	7T.
(17)	ANA D	F	4T.
(18)	ANA M	F R	7T
(19)	CMA	F	4T.
(20)	JMP 201SH	F R R	10T.

1000 JMP
1001 15
1002 20

(21) JP 1011H → cond^n sats, cond^n not sats. — FRR / FR* — 10T / 7T.

JP 1011
3000 JP
3001 11
3002 10
Total instr will take 10T

(i) 16 bit data transfer instruction

Imp (a) LXI Rp, 16 bit data.

→ IWS ≡ 3 Byte.

→ Rp ≡ BC ≡ B
DE ≡ D.
HL ≡ H
SP

→ operation: 16 bit data will store in Rp.

[Rp] ← 16 bit data.

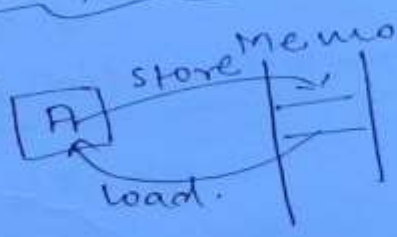
Ext LXI D 2059 H
[D] = 20H
[E] = 59H.

→ Add. mode: Immediate Add. mode.

→ M.Cy ≡ FRR ≡ 10 T.

Imp (b) LDA, 16 bit address

Note:-



→ IWS ≡ 3 Byte.

→ operation: [A] ← [16 bit add] in the memory

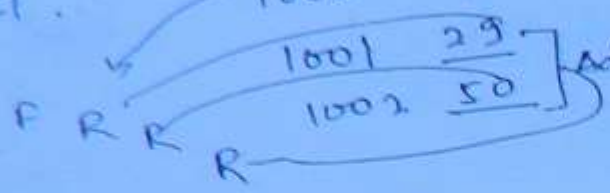
8 bit data available at 16 bit add. that is given in Instruction will load in accumulator.

Ext LDA 5029H
[A] = [[5029]]
[A] = 2DH

→ Add. Mode : Direct add. mode.

T-state

→ M-C. 1000 LDA ⇒ F R R R 13T.



42

(c) STA 16 bit add →

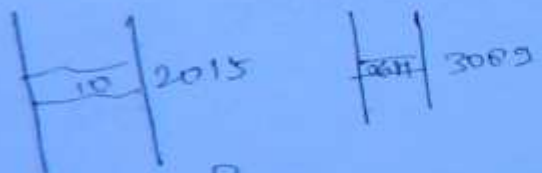
→ 2ws ≡ 3 Byte.

→ operation: [A] ⟶ [16 bit add]

content of [A] will store at the 16 bit add in memory that is given in the instruction.

→ Add. mode ≡ Direct Add. mode.

→ M.C ≡ F R R W ⟶ 13 T.



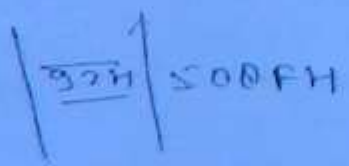
LDA 2015
STA 3009.

(d) LDAX Rp.

→ 2ws ≡ 1 Byte.

→ operation: 8 bit data available at the 16 bit add. in memory, that is given in the form of content of Rp, will load in Acc.

[BC] = 500FH.
LDAX B.
[A] = 92H



M-C. F R

→ Addl. Mode: indirect reg. addr. mode.

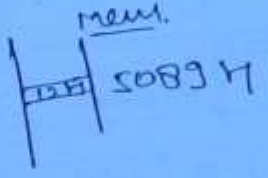
(c) STAX Rp.

→ Iws = 1 Byte.

→ operation: Content of accumulator will store at 16 bit addl. in memory, that is given in the form of content of Rp.

[A] → [[Rp]]

[A] = 12H.
LXI D 5089H
STAX D



→ m-cy :- FW.

Note: for LDAX, STAX.
Rp = BC
DE

→ Note: Data transfer inst., so status of flag will not affect.

6th Group.

(i) 16 bit arithematic instruction

(a) INX Rp.

→ Rp = BC / DE / HL → BD / H.

→ Iws = 1 Byte.

→ operation: Content of Rp will increase by one & result will store in Rp.

Rp ← Rp + 1 LSB.

→ Addl. Mode: Registry addl.

→ m-cy - S.

(b) DCX Rp :

→ 3WS = 1 Byte

→ Rp = BC
DE
HL.

→ operation:- Content of Rp decrease by one & result will store in Rp.

[Rp] ← [Rp] + 1

→ Add. mode :- Reg.

→ M- Cy :- S.

Note :- INX & DCX will not affect status of any flag.

7th Group (V. Imp)

(i) 8 bit logical rotational inst.

→ These inst. execute on the basis of content of accumulator.

→ When these inst. execute content of [A] will shift by 1 bit either left or right as per inst.

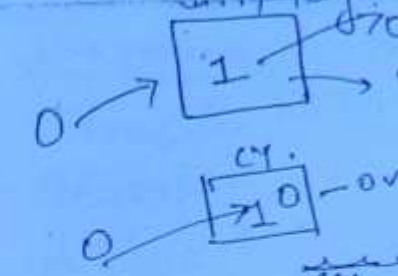
(a) RLC no. operand ; content acc. rotate left by 1 bit without Cy
(b) RAL no operand ; " " " " " " with "
(c) RRC no operand ; " " " Right " without Cy.
(d) RAR no operand ; " " " " " with Cy.

→ IWS :- 1 Byte.

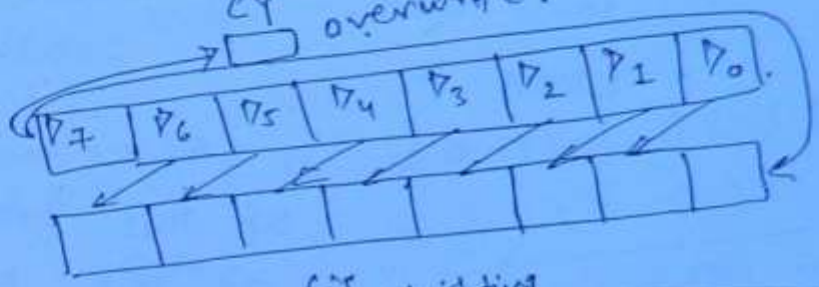
→ operation :- Content of [A] rotate left or right by 1 bit as per instruction.

→ Add. mode :- Implicit Add. mode.

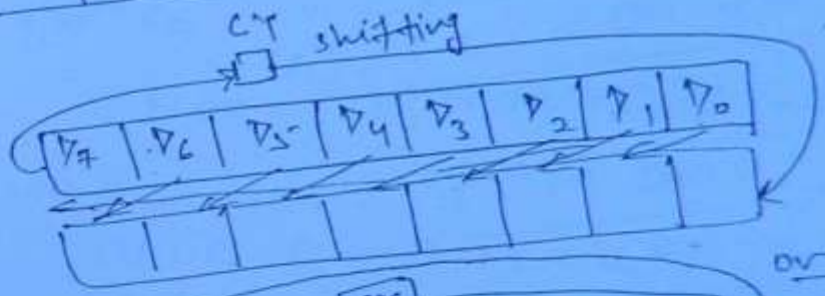
Note: ... swapping or swapping.



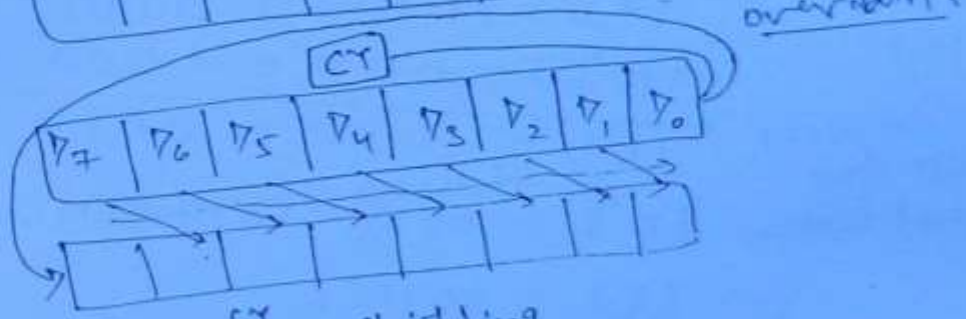
(i) RLC



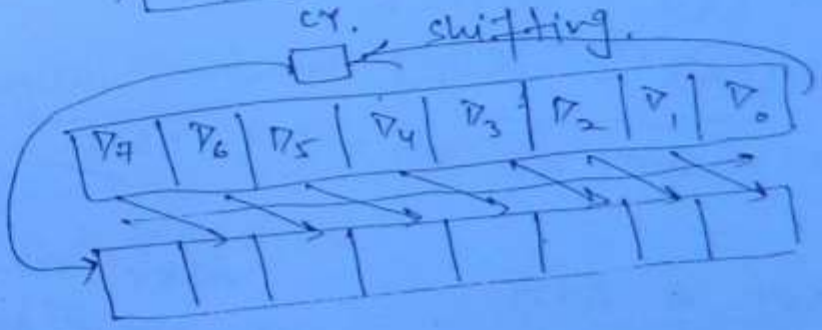
(ii) RAL



(iii) RRC



(iv) RAR



→ M-cy : Fetch (F).

Note :- Rotational inst. affect only one flag that is carry flag.

Q:

(46)

```
LXI H, 2000H
LDA 2002H
XRA M
MOV E, A.
MVI D, 20
LDAX D.
OUT 01.
```

memory add.	Content
2000H	00H
2001H	01H
2002H	02H
2003H	03H

After the execution of this program disphy at o/p port.

Sol: → [H] = 20H, [L] = 00H.

→ [A] = 02H.

→ [A] = 0000 0010

[M] = 0000 0000

[A] = 0000 0010

→ [E] = 02H, [A] = 02H.

→ [D] = 20H.

→ [A] = 02H.

→ disphy at o/p = [A] = 02H.

Q:

```
1000H: MVI A A1H
1002   LXI H 1007H
1005   SUB M
1006   OUT 05H
1008   HLT.
```

After the execution of above program disphy at o/p.

(a) A1H (b) 9CH

(c) 05H (d) Can't determine

Sol: 1000 → [A] = 1010 0001

→ [H] = 10, [L] = 07.

→ SUBM [A] = 1010 0001

[M] = 0000 0101

[A] = 1001 1100

Q: MVI A, F0H — How many time loop will execute.
ORA A
Loop. INR A
JNC Loop.
HLT.

Sol: → [A] = F0 = 1111 0000 → CY = 0
→ ORA → 1111 0000
[A] = 1111 0000
1

→ INRA [A] = 1111 0001 CY =
1111 0001
1
1111 0010 CY = 0

* INR will not affect flag

→ Infinite times loop will execute.

8th Group

(1) 8 bit logical compare instruction.

(a) CMP R.

→ R = A, B, C, D, E, H, L & M.
→ Size = 1 Byte.
→ operation = Content of 'R' will compared with con of accumulator & status of flag - affec accordingly.

Note: It is nothing but SUB (A-R) & status of flag will affect according the result of SU (A-R), but content of A & R will not change it means, Result of SUB (A-R) will discarded.

→ Addl. mode: R ≠ M = Reg. addl. mode = $\frac{M\text{-}C}{F}$
R = M = Indirect addl. mode = F R

→	CY	Z
[A] > [R]	0	0
[A] < [R]	1	0
[A] = [R]	0	1

⇒ Rest of flag according to the result of SUB (A-R).

48

(b) CPI 8 bit data.

→ Bus = 2 Byte.

→ operation :- 8 bit data compare with content of [A] & according to the result, status of flag will affect but content of [A] will not change.

→ Add. mode : Immediate Add.

→ M. Cy :- F, R.

→ All flag will affect.

	CY	Z
[A] > 8 bit data	0	0
[A] < 8 bit data	1	0
[A] = 8 bit data	0	1

⇒ Rest of flag according to the result of SUB (A - 8 bit data).

Note :- CMP, CPI will affect all flag.

Sign no. system.

① Direct sign mag repr.
② 1's compliment
③ 2's compliment.

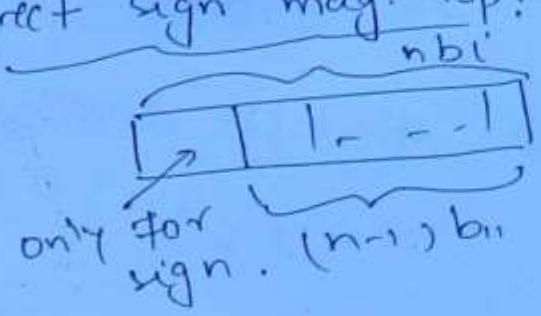
→ In all three one concept is common.

MSB always show the sign of no.

MSB = 1 ≡ -ve.

MSB = 0 ≡ +ve.

① Direct sign mag. repr.



+5 | 0 | 1 | 0 | 1 |

-5 | 1 | 1 | 0 | 1 |

② 1's compliment repr.
③ 2's compliment reprs. }.

→ in that two concept is common.

(i) +ve no. always written as straight binary with MSB = 0, either in 1's comp. or 2's comp

Ex's { given no. in 1's comp. find it's decimal eq

0 1 0 0 = +4
0 1 1 0 = +6.

given no. in 2's comp. find it's decimal eq. —

0 1 0 0 = +4
0 1 1 0 = +6.

(ii) If we take 1's compliment or 2's compline of any binary no. then its sign will be cha but magnitude will be same.

Ex: write -6 in 1's compliment.

+6 ≡ 0 1 1 0 $\xrightarrow{\text{1's comp}}$ 1 0 0 1 ≡ -6.

in 2's compliment.

+6 ≡ 0 1 1 0 $\xrightarrow{\text{2's comp.}}$ 1 0 1 0 = -6.

1 - while 1st one take

Ex: ① Given no. is in 1's comp. Find its decimal eq.

1010 = −(+5) = −5.

1's ↓

0101

= 5

② Given no. in 2's comp. Find its decimal eq.

1010 $\xrightarrow{\text{2's comp.}}$ 0110 = −(6).

Experiment

* [A] = 37 H. ⇒ [A] = 0 0 1 1 0 1 1 1

[B] = 25 [B] = 0 0 1 0 0 1 0 1

[A] − [B]. = 0 0 0 1 0 0 1 0

[A⁻]

CY = 0
S = 0
Z = 0
P = 1
AC = ?.

In computer.

A + (−B)

A + (2's B).

[A] = 0 0 1 1 0 1 1 1

2's [B] = + 1 1 0 1 1 0 1 1

0 0 0 1 0 0 1 0

CY = 1
S = 0
Z = 0
P = 1
AC = 1

* [A] = 25 H [A] = 0 0 1 0 0 1 0 1

[B] = 37 H [B] = − 0 0 1 1 0 1 1 1

1 1 1 0 1 1 1 0

CY = 1
S = 1
Z = 0
P = 1
AC = ?

In computer

[A] = 0 0 1 0 0 1 0 1

2's of [B] = + 1 1 0 0 1 0 0 1

1 1 1 0 1 1 1 0

CY = 0
S = 1
Z = 0
P = 1
AC = 0

1st method.

① (A-R) direct. CY, S, Z, P } use this.

A + (2's of R) → AC. } use this.

② nd method

A - B = A + 2's. B. CY = $\overline{CY}$ }

S, Z, AC, P }

Example:

```
MVI A 29
ORA A.
RLC
RAL.
HLT.
```

After the execution of prog: content of A = ? & CY = ?

Soln → [A] = 29 = 0010 1001

→ ORA A.

0010 1001

0010 1001 → CY = 0

[A] = 0010 1001

→ RLC

CY = 0 without carry.

[A] = 0 0 1 0 1 0 0 1 → CY = 0.

0 1 0 1 0 0 1 0

[A] = 52H, CY = 0.

→ RAL

CY = 0 with carry.

[A] = 0 1 0 1 0 0 1 0

[A] = 1 0 1 0 0 1 0 0

→ [A] = A4, CY = 0.

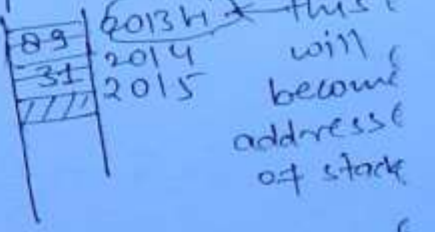
* STACK

→ Stack is the group of continuous memory locati in main memory that is used for temporary storage of information during the execution of main program.

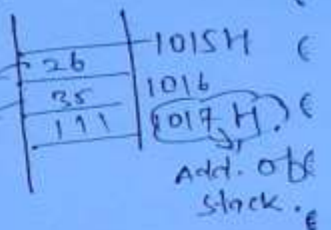
(S2) 2012 / 2013 or 2019 / 201A.

→ At a time two byte data can store at the top of stack, or two byte data can retrieve from the top of stack.

→ If two byte data store at the top of stack, it grows upward with numerically decreasing order of its address.



→ If two byte retrieve from the top of stack it goes downward with numerically increasing of its address.



→ In 8085, two instructions are defined to store or retrieve the two byte data from the top of stack, i.e. PUSH, POP.

PUSH = store
POP = Retrive.

→ Stack can be initialize in main memory by inst; LXI SP 16 bit data.

* SP = 16 bit data ⇒ LXI SP 5089H
SP = 5089H

→ During the execution of CALL subroutine, address of next instruction will store at the top of stack.

→ Stack works on the principle of LIFO ≡ Last in (i/p) first out (o/p). (53)

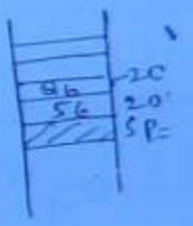
* (i) PUSH Rp.

→ Rp ≡ BC ≠ B.
DE ≠ D.
HL ≠ H.
PSW = ACC flag Reg.

→ Bws; 1 Byte.

→ operation; When this inst. will execute, the content of register pair will store at the top of stack.

Eg: LXI SP, 2019H → SP = 2019H
LXI B, 5686H → [BC] = 5686 (B C)
PUSH B.
MVI AC.
A = 06.



→ After the execution of PUSH instruction, cont of S.P. will decrease by two.

→ Add. mode ≡ Reg. add mode ← source data — PUSH
2nd ← dest. data
Reg. mode.

→ M-cy. = 1 or 3 PUSH B
3 WW.

Note: ① It is data transfer instruction. So status of flag will not affect.
② There is no conditional PUSH instr.

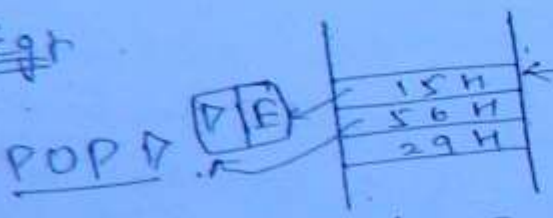
(ii) POP Rp :

→ Rp = BC → B
DE → D
HL → H
PSW → Acc. flag.

→ Bws = 1 Byte.

→ operation : When this instr. will execute, content of two byte data will retrieve from the top of stack & store in Rp.

Eg:



D = 56H
E = 15H. } SP = 2018H.

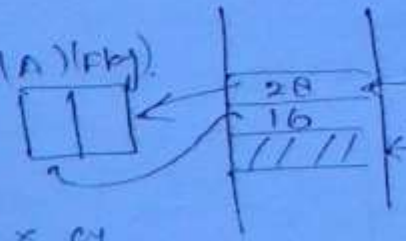
→ After the execution of POP instr., content of S.P. will increase by two.

→ Add. mode :- POP Rp → source data = Indirect Reg. add mode
→ destination data = Register add. mode

→ M-Cy :- 1 F RR.

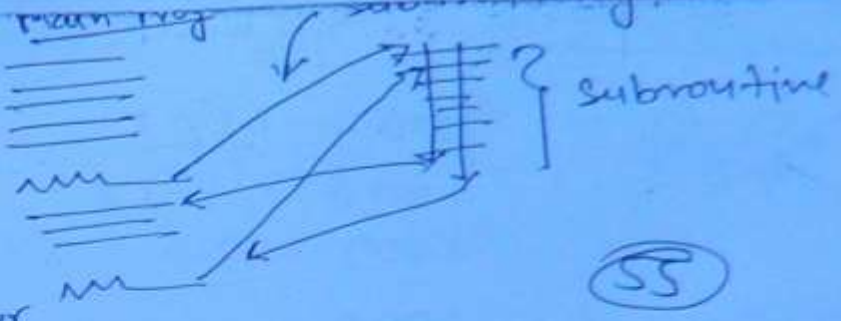
Note: POP is data transfer inst, so status of flag will not affect, but in case of POP PSW, status of flag may be affected indirectly.

S | Z | Ac | P | Cy
1 | 0 | 0 | 1 | 1



POP PSW. [A] = 16H
[F] = 28H = Z ∨ AC ∨ P ∨ CY
= 0 0 1 0 1 0 0 0

* Subroutine :-



→ It is set of inst. that written seperately from main program regarding the task occur in main program frequently.

→ To develop subroutine in 8085 two instructn are define.

(i) CALL.
(ii) RET.

(i) CALL inst :-

It is of two types -

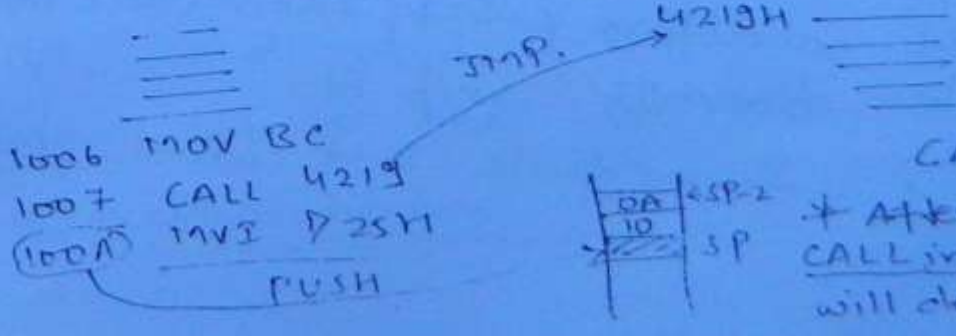
(a) Unconditional CALL inst.
(b) conditional CALL inst.

(a) Unconditional CALL inst :-

* CALL, 16 bit add

→ I ws = 3 Byte.

→ operation : When this instruction will execute control/execution will transfer at 16 bit vector location given in the instruction but befo transfer, address of next instruction will stove at the top of stack.



CALL = PUSH + JMP

* After the execution of CALL inst content of S will decrease by two.

→ Add. mode :- Immediate add. mode.

→ M-Cy :- 1007 CAL . S R R WW ≡ 18T.
1008 19
1009 42 .
100A
PC →

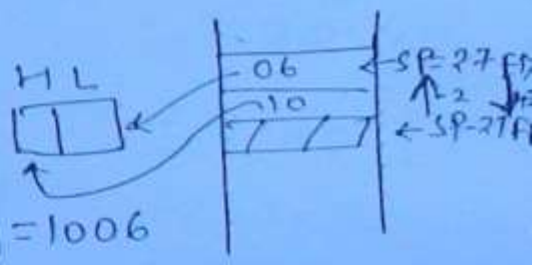
(56)

Example → 1000H : LXI SP 27FFH
1003 : CALL 1006H
1006 : POP H

After the execution value of SP & HL Pair.

	S.P.	HL
(a)	27FD	1006
(b)	27FD	1003
(c)	27FF	1006
(d)	27FF	1003

→ SP = 27FFH



(b) Conditional CAL inst :-

CC 16 bit add.; CALL inst. will execute if CY=1.
CNC " " ; " " " " " CY=0
CZ " " ; " " " " " Z=1
CNZ " " ; " " " " " Z=0
CP " " ; " " " " " B=0
CM " " ; " " " " " S=1
CPE " " ; " " " " " P=1
CPO " " ; " " " " " P=0.

→ 3WS ≡ 3 Byte.
→ operation :- If condition satisfy then CALL inst. will execute otherwise skip it.
→ Add. mode : Immediate Add. mode.

Ex:-

1005 CZ 2068 H
100B MOV BC

1005 CZ → Satisfy → S R R W W
ST { 1006 6B
1007 20 → not satisfy → SR
1008 MOV BC

Note:- * CALL inst. is data transfer inst., no status of flag will not affect.

* 6th group (Regarding).

16 bit arithmatic inst.

(iii) DAD Rp.

→ 2WS = 1 Byte.

→ Rp = BC, DE, HL.

→ Operation:- Content of Rp will get added in HL pair & result will stored in HL.

[HL] ← [HL] + [Rp]

→ Add. mode = Reg. add. mode.

→ M-cycle = F B B

* CALL (unconditional & conditional CALL insatisfy case) is largest inst of 8085.
≡ 18T.

Date 3/11/12

(ii) RET.

It is of two type -

(a) Unconditional RET.
(b) Conditional RET.

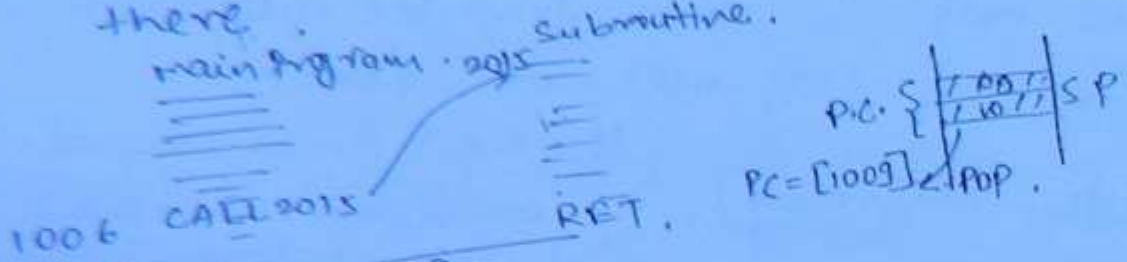
Note:- It is the last inst. of every subroutine

(a) Unconditional RET.

* RET no operand

→ BWS = 1 Byte

→ operation: When this inst. will execute, two byte (latest stack) data retrieve from the top of stack, store in P.C. & next inst. will fetch from there.



RET ≡ POP + JMP.

→ Add. mode: → Implicit Addl. mode.
↳ Reg. indirect Addl. mode. [S.P.]

→ M-C:- S RR.

Note: This is data transfer inst. so status of flag will not affect.

→ After the execution of RET inst. content of S.P. will increase by two.

Ex's LXI SP 2715
CALL 3000H

After the execution of this program content of S.P will be.

→ 3000H LXI H 1015H
PUSH B
PUSH PSW
PUSH H
LXI SP 3CF4H
POP H
POP PSW
POP B
RET.

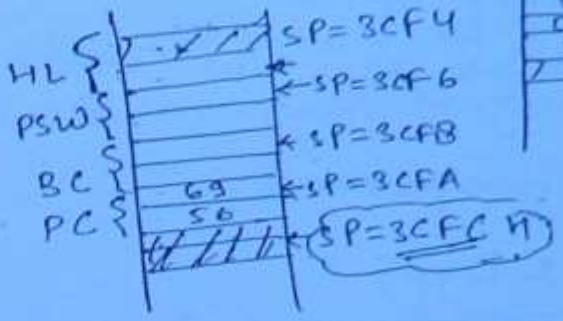
Soln → SP = 2715 H

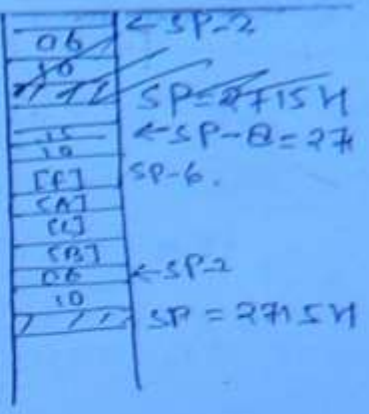
→ 1006 ...

→ [HL] = 1015 H

SP ≡ 3CF4H.

(59)





(b) Conditional RET inst :-

RC no operand ; RET inst will execute if CY=1.
RNC " " ; " " " " " CY=0
RZ " " ; " " " " " Z=1
RNZ " " ; " " " " " Z=0
RP " " ; " " " " " S=0
RM " " ; " " " " " S=1
RPE " " ; " " " " " P=1
RPO " " ; " " " " " P=0.

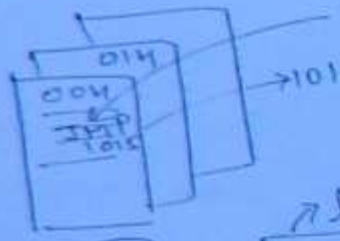
→ IWS ≡ 1 Byte.
→ Operation : If condition satisfy then RET will execute, otherwise skip it.
→ Addl. mode : → Indirect Reg. mode.
→ Implicit Addl mode.
→ M-CY → cond satisfy. → S R R
cond. not satisfy → S/F*

Note :- → It is data transfer inst. so status of flag will not affect.

RESTART =>

→ It is just like one byte CALL inst.
→ This inst. is used when execution trap among the interrupts.
→ It is like as s/w interrupt.

* RSTN nooperand
→ 1WS = 1 Byte.
→ N = 0, 1, 2, 3, 4, 5, 6, 7.
→ Operation: When this inst. will execute, content control (execution) will jump at the specific vector location of memory page no. - 00H.



Page No. XX YYH ← line

00YY
00YY.

Note: xx is the hexadecimal conv. of HXB.

16 | 16 | 0
1

$(16)_{10} = (10)_H$.

Inst.	Vector location
RSTN	00 XX
RST 0	00 00 H
RST 1	00 08 H
RST 2	00 10 H
RST 3	00 18 H
RST 4	00 20 H
RST 5	00 28 H
RST 6	00 30 H
RST 7	00 38 H

→ Add. mode = Implicit addl. mode.
→ M-Cy = 5.

Ex:

main Prog →

```
1000: MOV BC
1001: CALL 5029H
1004: MOV A,E
```

Subroutine →

```
5029: MVI A 00H
502B: CALL 502EH
502E: INR A
      RET
```

[A] = ?

(61)

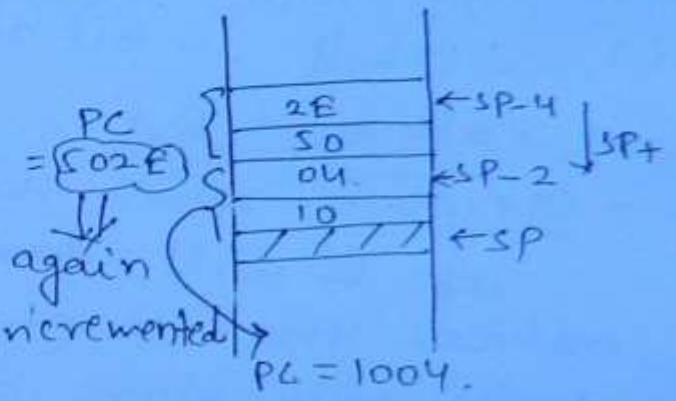
After the completing the execution of subroutine when control comeback in main program, at this time value of [A] = ?

Sol'n → [A] = 00H

= 01H.

RET → will Pop from top of stack.

i.e. (502E).



[A] = 01

+ 1

[A] = 02H

9th Group

* Instruction Related to HL Pair.

(1) LHLD 16 bit add.

→ 3ws ≡ 3 Byte.

→ operation : When this inst. will execute, 8 bit data available at the memory location the is given in the inst. load in L & next 8 bit data at next memory location will load in H.

[L] ← [16 bit add]

[H] ← [16 bit add+1].

Eg: LHLD 2019

[L] ← [[2019]] => [L] = 27H

[H] ← [2020] => [H] = 60H.

→ Add. Mode : Direct add. mode.

→ M-C cycle : - F RR RR → 16T.

100H LHLD 2019.

⇒ 1000 (LHLD).
1001 R (19H)
1002 R (20H).

(62)

(ii) SHLD 16 bit add

→ BWS = 3 Byte.

→ Operation : When this inst. will execute content of 'L' will store at the 16 bit memory address that is given in the inst. & content of 'H' will store at next memory address.

[L] → [16 bit add.]
[H] → [16 bit add+1].

→ M-CY : F RR WW → 16T.

1006 SHLD 2015

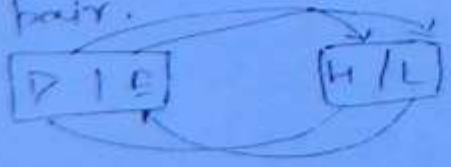
1006 (SHLD)
1007 R (15)
1008 R (20).
W: 2015
W: 2016

→ Add. Mode = Direct add. mode.

(iii) XCHG no operand.

→ BWS = 1 Byte.

→ Operation : When this inst. will execute content of HL pair exchange with content of DE pair.



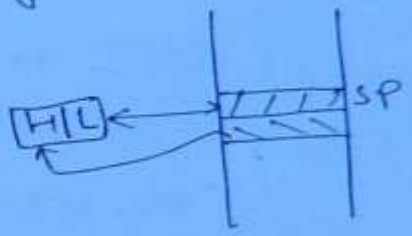
→ Add. mode → Implicit add. mode.
↳ Register add. mode.

→ M-Cy ≡ F.

(iv) XTHL no operand.

→ 2WS ≡ 1 Byte.

→ Operation: When this inst. will execute content of HL pair will exchange from top of stack.



→ For intermediate storage, memory locations are used

XTHL ≡ PUSH + POP.

→ M-Cy: FWWR RR.

→ Add. mode → Implicit
→ Register.
→ Indirect Reg.

(v) SPHL no. operand.

→ 2WS ≡ 1 Byte.

→ Operation: When this inst. will execute content of HL pair will copy in S.P.

[SP] ← [HL]

Note: It is indirect method to initialize the stack in main memory.

→ Add mode → Implicit
↳ Regis. add.

→ M-Cy → 5

LXI SP 2015
1001 CALL 3019 H
1004 —
3019 —
SPHL

SP=

(vi) PCHL nooperand

→ 3ws ≡ 1 Byte.

→ Operation: After the execution of this inst.; content of HL pair will copy in P.C.

→ M-Cy ÷ S

→ Add. mode: → Implicit.
→ Reg Add.

Eg: 1001 LXI H 3125 H
1004 MVI A, 29 H
1006 PCHL
1007 —

After the ex. of PCHL ins. nex. will fetch from.

Sol: → [HL] = 3125.
→ [A] = 29
→ [PC] = 3125.

Note: All above inst. are data transfer inst, so status of flag will not affect.

10th Group.

8 bit advance arithematic inst.

(i) ADC R

→ R ≡ A, B, C, D, E, H, L & M.

→ 2ws ≡ 1 Byte.

→ Operation: When this inst. will execute content of 'R' will get added in [A] with carry flag status & result will store in [A].

[A] ← [A] + [R] + Cy. LS B.

Eg: MVI A 25H [A] = 25H
MVI B 19H [B] = 19H.
ORA A [A] = 25H. CY = 0.
A B C D

[A] = 0 0 1 0 0 1 0 1
[B] = 0 0 0 1 1 0 0 1
0
[A] = 0 0 1 1 1 1 1 0 → CY = 0
→ AC = 0
→ S = 0
→ Z = 0
→ P = 0
[B] = 19H.

→ Add. Mode: R ≠ M ≡ Reg. add ≡ F.
R = M ≡ Indirect ≡ F.R.

(ii) ACI 8 bit data.
→ Bws ≡ 2 Byte.
→ Operation: 8 bit data will get added with [A] with Carry flag status & result will store in [A].
[A] ← [A] + 8 bit data + CY LSB.
Eg:
→ Add. mode: Immediate add. mode.
→ M - CY ≡ F R.

(iii) SBB R
→ Bws ≡ 1 Byte; R ≡ A, B, C, D, E, H, L & M.
→ Operation: Content of 'R' will get substract from [A] with carry flag status & result will store in [A].
[A] ← [A] - [R] - CY LSB.
→ Add. mode: Reg. Add. mode.
→ M. CY: R ≠ M → F
R = M → F.R

(iv) SBI 8 bit data

operation: [A] ← [A] − 8 bit data − [Cy]

→ IWS → 2 Byte.

→ Add. Mode: Immediate.

→ M-Cy. : FR.

Note: All above four inst. ADC, ACI, SBB, SBI is arithmatic inst. so status of all flag will affect.

11th Group

Some advance inst.

(i) STC no operand

→ IWS = 1 Byte

→ Operation: After the execution of this inst, status of carry flag will get set, regardless of previous status.

→ Add. mode: Implicit add. mode.

→ M-Cy: F

(ii) CMC no operand

→ IWS = 1 Byte.

→ operation: After the execution of this inst, status of Carry flag will get complement,

$CY \leftarrow \overline{CY}$

→ Add. mode: Implicit add. mode.

→ M-cycle: F.

Note: STC & CMC affect only one flag that is carry flag.

(iii) DAA no. operand.

→ Decimal adjustment of content of ACC.

→ This :- 1 Byte.

→ Operation :- When this instr. will execute conten of [A] will adjust in BCD format by assumi earlier operation was BCD addition.

→ Add. mode :- Implicit add. mode.

→ m-cy :- F. / Note :- DAA inst. affect status of all flag.

Note :- ① If lower nibble of [A] is greater than 1001, 0110 will get added in it.

② If L. nibble of [A] is 1001 or less than 1001, but A.C = set, then 0110 will get added in it.

③ If U. nibble of [A] is greater than 1001, 0110 wi get added in it.

④ If U. nibble of [A] is equal to 1001 or less tha 1001, but cy = set then 0110 will get added in it.

Ex's :-

```
LXI H 0A79H
MOV A,L
ADD H
DAA
MOV H,A
PCHL.
```

After the ex. of PCHL inst. next ins. will fetc from.

(a) 6019H
(b) 0379H
(c) 6979H
(d) None of these.

Soln [H] = 0A79H

[A] = 79

ADD H =>

	0	1	1	1	1	0	0	1
	1	0	0	0	1	0	1	0
[A] =	0	0	0	0	0	0	1	1
	0	1	1	0	0	1	1	0
→DAA:-[A] =	0	1	1	0	1	0	0	1

9 H

S	Z	AC	P	CY
0	0	1	1	1

0 0 0 1 0

→ [H] = 69H.

PC ← [HL]

(PC = 6979H)

INTERRUPT

In 8085 five H.W. interrupts are present.

- TRAP → Non maskable.
- Maskable:
  - RST 7.5
  - RST 6.5
  - RST 5.5
  - INTR.

Note: TRAP also known as RST 4.5.

→ On the basis of different characteristics interrupts are classified in different groups.

(i) Maskable & Non-maskable interrupt

Interrupts that can make disable ≡ Maskable int.
Interrupts that can not make disable ≡ Nonmaskable

Note → To control interrupt process in 8085 a interrupt enable flip-flop is present.

→ If interrupt enable ff is set ≡ interrupt process enable.

→ If int. enable ff. is reset ≡ interrupt process is disable.

Note: Masking & Non-masking concept valid only when interrupt process is enable.

→ To set or reset of interrupt enable ff. two inst. are defined

(a) EI no operand.

Ins ≡ 1 Byte.

operation: → interrupt enable ff will get set.
→ interrupt process will enable.

Addl mode ≡ Implicit addl.

M-cy. ≡ 1.

(b) DI no operand.

→ IM1S = 1

→ Operation :- Interrupt enable F.F. will get reset
→ Interrupt process will disable.

→ Add. mode : Implicit.

M-C/ : F.

Note ⇒ * EI & DI are the m/c C/ control inst so status of flag will not affect.

*Note ⇒ TRAP is independent of EI & DI.

(ii) Vectored & Non-vectored interrupt.

When vectored interrupt acknowledge control execution will jump at fixed vector location of memory page 00H.

TRAP
RST 7.5
RST 6.5
RST 5.5
} → vectored.

INTR → Non-vectored.

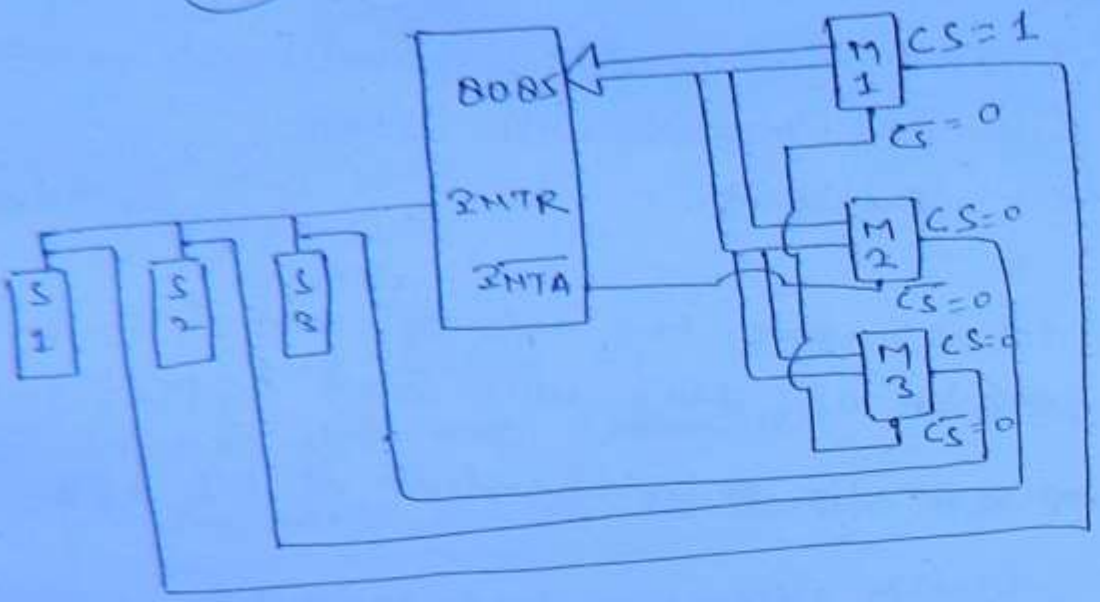
vector location to non vectored interrupt provid by externally.

Eg. vectored int.

TRAP
RST 7.5
RST 6.5
RST 5.5

Vector location

0024H
003CH
0034H
002CH



S_1, S_2, S_3 } sensors

M_1, M_2, M_3 } memory

* Triggering

Edge ← TRAP } Edge trigger.
level both RST 7.5

RST 6.5
RST 5.5
INTR. } Level trigger.

* Priority.

TRAP — highest
RST 7.5
RST 6.5
RST 5.5
INTR. — lowest.

Trap has the highest priority, INTR has lowest priority.

Note → when an interrupt is acknowledge following steps execute automatically.

(i) Execution of current inst. will complete first.
(ii) Add. of next inst. will store at the top of stack.
(iii) PC inst. will execute automatically.
(iv) Execution will transfer at interrupt sub routine.

Note → ① Programer should write EI inst. at the last of interrupt service routine.

Note → ② In externally initiated signal HOLD has the highest priority.

Note → ③ Min Time duration for which an interrupt should occur to get definite execution.

≡ 18T.

≡ 17.5 T.

* SIM (Set interrupt mask)

→ This inst. is used for.

* to mask the interrupt.
* to serial transfer of data through SOD pin.

Note: This inst. execute on the basis of content of acc.

→ SIM no operand

→ BWS ≡ 1 Byte.

→ Operation: On the basis of accumulator this inst. execute as per their property.

→ Addl mode: Implicit Add.

→ M-Cy: 1.

[A] =

D7	D6	D5	D4	D3	D2	D1	D0
SOD	SOE	X	R7.5	IE	M7.5	M6.5	M5.5

1 ← serial data transfer through SOD Pin.

1 → Serial data transfer process will Enable.

1 ← RST 7.5 to be Reset.

1 → Interrupt Enable F.F to be set.

1 → RST 5.5 to be mask

1 → RST 6.5 to be mask

1 → RST 7.5 to

Ex: MVI A, 0EH. [A] = 00001110
SIM.

* RIM (Read Interrupt Mask)

→ This interrupt inst. is used for.
* To known status of pending interrupts.
* To know status of masked inter.
* Used for serial data transfer through SID pin.

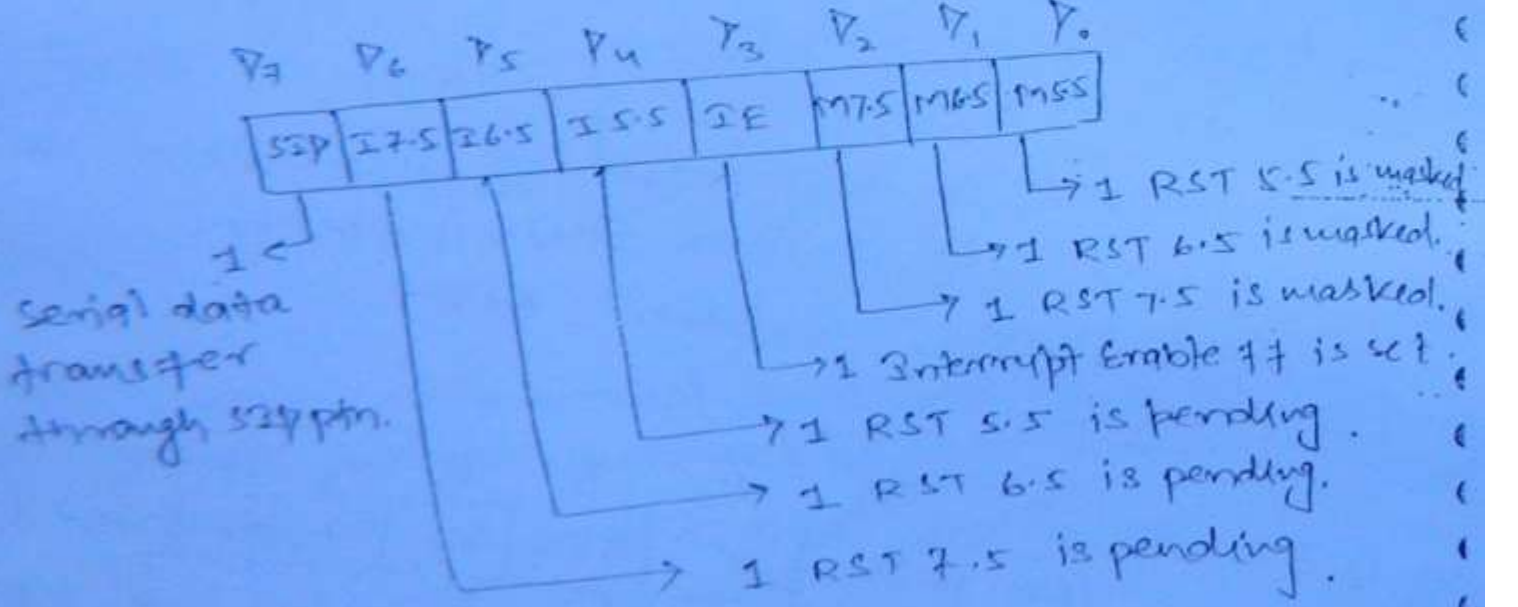
RIM no operand

→ Bws = 1 Byte.

→ Operation: When this inst. execute according its feature information will load in [A].

→ Add. Mode: Implicit

→ M-Cy: 1.



Ex: RIM
OUT 0bit port = seven sig. displ.

Note: RIM & SIM is m/c control inst. so status of flag will not affect.

Q:
```
EI
RIM
ANI 08H
SIM
```

[A] = D7 D6 D5 D4 D3 D2 D1 D0

08 = 0 0 0 0 1 0 0 0

[A] = 0 0 0 0 1 0 0 0

What kind of task is performed by above set of inst.

(a) send bit out on SOD pin

(b) Accept bit in from SID pin

(c) Accept RST 7.5 interrupt

(d) Reset RST 7.5 interrupt.

COMV - 15 marks.

(i) How many times µp execute memory read m/c cycle = ?

(ii) How many times µp perform memory read operation or how many times data read from memory.

(iii) How many times µp read operation execute or how many time $\overline{RD}$ signal will go active low.

(iv) Total T-state req'd

(v) Total time duration of execution or time elapse in the execution of prog

f = 3MHz.

Q:

```
LXI H 2041H
MVI A, 06H
CMP M.
MOV B, A
IN 20H
SUB B
OUT 52H
INX H
MOV M, A
HLT.
```

10T = FRR
7T = FR
7T = FR
4T = F
10T = FR3
4T = F
10T = FRO
6T = RS
7T = FW
4T = F

Sol: (i) 6 times. (ii) 16 times. (F+R). (iii) 17 times

(iv) 69 T. (v) $T = 69 \times \frac{1}{3} \times 10^{-6} = 23\,\mu sec$.

Q: MVI B, 28H → FR=7T

LOOP NOP → F = 4T

DCR B → F = 4T

JNZ loop → FRR/FP = 10T/7T

HLT. → F = 4T.

T = 0.5 µsec.

total times elapsed/Req =

$28H = (40)_{10}$

Suppose, B = 01

$\frac{1}{00}$ } Z = 0, 1

B = 03 { 3 times.

$\Rightarrow$ Total T = 7T + 39[10T] + 1[15T] + 4T.

= 720 T.

Q:

```
      MVI B  38H   ≡ FR = 7T
loop2 MVI C  FFH   ≡ FR = 7T
      DCR C        ≡ F  = 4T
loop1 JNZ loop1    ≡ FRR/FR = 10T/7T
      DCR B        ≡ F  = 4T
      JNZ loop2.   ≡ FRR/FR = 10T/7T
      HLT.         ≡ F  = 4T.
```

op. f = 2MHz.

Soln $(38)_H = (56)_{10}$.

$(FF)_H = (255)_{10}$.

Total 'T' for inner comp. 255 times.

= 254(14T) + 1(11T) = 3567 T.

Total 'T' for program = 7T + 55[7T + 3567T + 4T + 10T]

+ 1[7T + 3567T + 4T + 7T,

+ 4T.

= 200936 T.

INTERFACING

- memory interfacing.
- I/O interfacing
  - memory map I/O interfacing.
  - I/O mapped I/O interfacing or Peripheral mapped I/O interfacing

Common steps of interfacing-

→ All command signal connect directly.

→ Data pins connect directly as.

$(MSB)_{device} \longleftrightarrow (MSB)_{MP}$.
:
$(LSB)_{device} \longleftrightarrow (LSB)_{MP}$ } data pins

→ Addl. pins of devices connect directly as

$(LSB)_{device} \longleftrightarrow (LSB)_{MP}$.
:
$(MSB)_{device} \longleftrightarrow (---)_{MP}$ } Addl. pins.

→ Remaining Address buses used for develop-ent of chip selection logic.

Note→ chip selection logic can be developed by two ways-

(a) By use of logic gates or buffer ckt.

(b) By use of decoder ckt.

Memory Interfacing

Q. Let → Memory 4K Byte. = $2^{12} \times 8$.

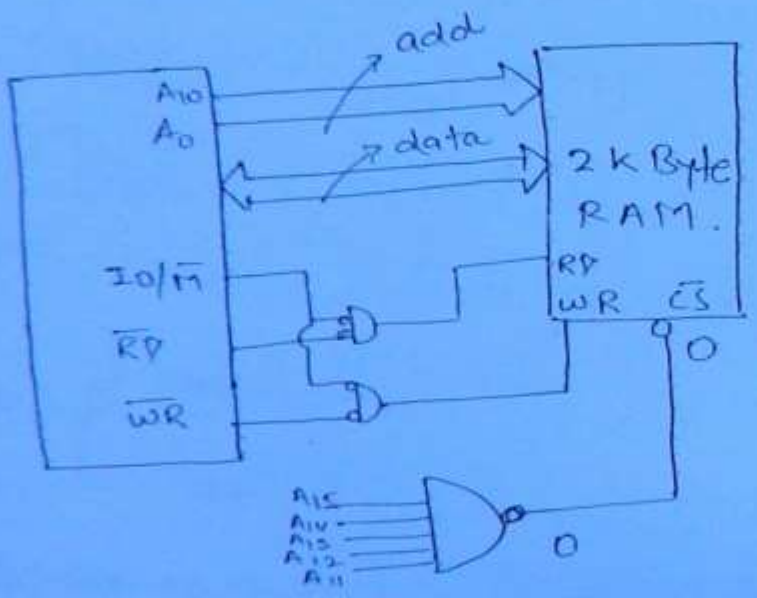
$2^n \times d$. $n = 12 \Rightarrow$ Add. pins

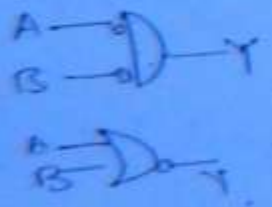
$d = 8 \Rightarrow$ Data Pins

A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_0	
0	0	0	0	0	0	0	0	0	0	0	= 000H }
1	1	1	1	1	1	1	1	1	1	1	= 111H.]

Note: First find address & data pin orling of memory.

Q: Find out memory add. range interfaced with 8085 MP.





$Y = \bar{A} \cdot \bar{B}$

$= \overline{A+B}$

Sol: 2K Byte. | Add = 11

$2KB = 2^{11} \times 8$ | Data = 8.

for chip selection

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	1	1	1	1	1	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
min add =	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

min add ≡ F800H
may add ≡ FFFFH
→ That means b/w this if process gives value, memo is selected.

(77)

Q: 4K Byte RAM, interfaced with 8085 µP. with chip selection logic, $CS = \overline{A_{15}} \cdot A_{14} \cdot A_{13}$. Then find its memory interfaced range.

(a) 5000H to 5FFFH
(b) 6000H to 6FFFH
(c) 6000H to 6FFFH & 7000H to 7FFFH
(d) 5000H to 5FFFH & 6000H to 6FFFH.

Sol'n 4K Byte $= 2^{12} \times 8$.

Add. = 12.
Data = 8.

$CS = \overline{A_{15}} \cdot A_{14} \cdot A_{13}$

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	1	1	X	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

X=0

min add = 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
= 6000H

max add = 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
= 6FFFH

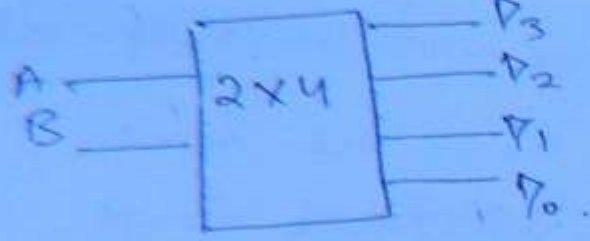
X=1

min = 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
= 7000H

max = 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
= 7FFFH.

Chip selection logic by decoder kit

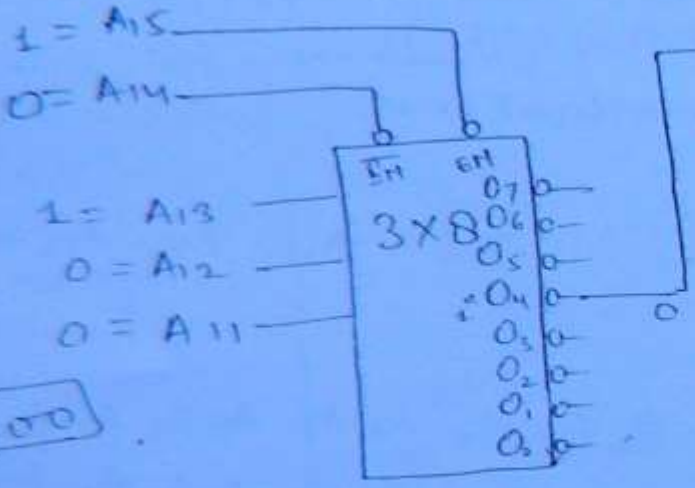
decoder



AB	Y_3	Y_2	Y_1	Y_0
0 0	0	0	0	1
0 1	0	0	1	0
1 0	0	1	0	0
1 1	1	0	0	0

$$Y(AB) = \bar{A}\bar{B}Y_0 + \bar{A}BY_1 + A\bar{B}Y_2 + ABY_3.$$

Q1



Find out memory add. range of interfaced memory.

Sol'n $2K\,Byte = 2^{11} \times 8$

Add = 11, Data = 8.

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	0	1	0	0	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

min. add = A000H.

1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	0

max. add = A7FFH.

Case-1 - If decoder I/P is not given.

Higher order bus at ≡ MSB of decoder I/P.

Lower order bus at ≡ LSB of decoder I/P.

Case-2 O/P of decoder → I/P of decoder → Add. Bus Value.

① I/P (A B C) → A = A_{13} = 1
1 0 0 B = A_{12} = 0
C = A_{11} = 0.

② I/P (C B A) → O/P
1 0 0 A = A_{13} = 0.
B = A_{12} = 0.
C = A_{11} = 1.

Date 10/12/11

I/O Interfacing

Comparison.

Characteristic	Memory mapped I/O int.	I/O mapped I/O interface
① Command sig.	MEMRD / MEMWR	IORD / IOWR
② Instructions.	All memory related inst. are valid.	IN & OUT only two inst are valid
③ Execution	Data transfer b/w I/O & any memory reg, A & L operation perform directly with any reg.	Data transfer b/w Acc & I/O is possible only.
④ No. of device interface.	64 K Byte memory shared b/w system memory & I/O devices	Max^{m} 256 I/P & 256 O/P dev can be interface
⑤ H/w requirement	More H/w required	Less H/w reqd.
⑥ Speed	Slower	Faster.
⑦ Application	Smaller System	For longer System

Some important peripherals

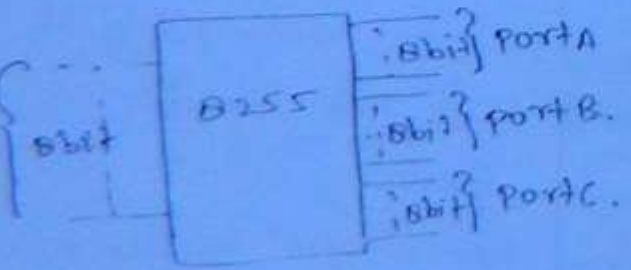
8255 ≡ Programmable peripheral interface.
8237 ≡ DMA controller.
8279 ≡ Programmable keyboard & display interface.
8259 ≡ Programmable interrupt controller ckt.
8155 ≡ Programmable I/O port & Timer ckt.
8254/8253 ≡ Programmable interval timer.

Some important digital IC PSU.

74182 ≡ Look ahead carry generator.
74180 ≡ 8 bit parity generator & checker ckt.
7493 ≡ 4 bit binary counter
7447 ≡ Seven segment decoder.
7490 ≡ decade counter.
7400 ≡ Quad 2 I/P NAND gate.
7402 ≡ " " " NOR "
7408 ≡ " " " AND "
7432 ≡ " " " OR "
7486 ≡ " " " EX-OR "

8255 (PPI)

→ It is 40 pin IC.



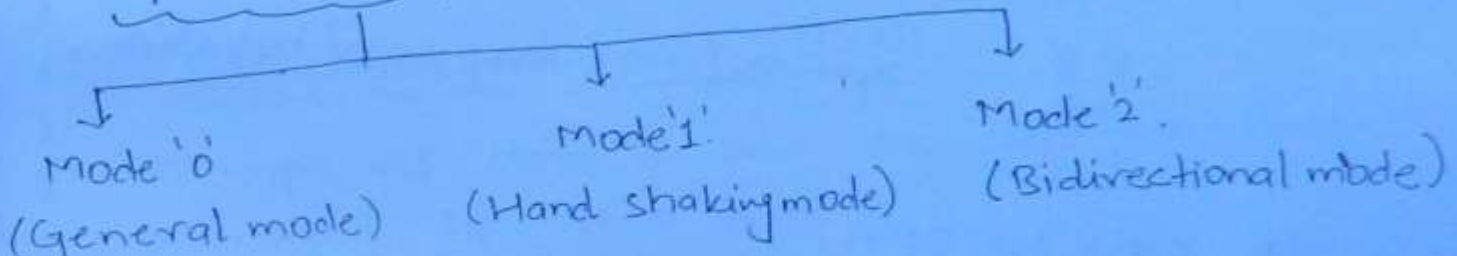
→ It has three port each having port add. of 8 bit port A, port B, port C.

→ Every port individually can act as I/O port.

→ Port 'C' can also works as two port as port 'C' upper & port 'C' lower having port add. of 4 bit (lower nibble & upper nibble). (81)
& port 'C' upper & port 'C' lower generate control signal for port 'A' & port 'B' respectively.

→ It works in two mode.
(i) Bit set Reset mode (BSR)
(ii) I/O mode.

I/O mode



(i) Mode '0' or General mode → In this mode port 'A', port 'B', port 'C' works as I/O mode seperately.

(ii) Mode '1' or Handshaking mode → In this mode Port 'A', Port 'B' works in hand shaking mode & Port 'C' upper & Port 'C' lower generate control sig respectively.

(iii) Mode '2' Bidirectional mode → In this mode of operation Port 'A' works in bidirectional mode only & port 'C' upper generate cont sig for it.

W.B. Page-93.
Questⁿ ⇒ 22.

ors 7 page 93:

9.22

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	1	1	1	1	✗	1	1
1	1	1	1	1	X	0	0

Case 1 X=0 1 1 1 1 1 0 0 0 F8H to

X=0 1 1 1 1 1 1 0 0 FBH

X=1 1 1 1 1 1 0 1 1 FCH to

X=1 1 1 1 1 1 1 1 1 FFH

F8 ⇒ This is in continuation.
+1
FC

⇒ F8H to FFH.

8086 (Introduction)

→ Comparision b/w 8085 & 8086.
→ Internal Architecture.
→ Memory segmentation & Physical add.
→ Flag Reg.
→ Address Mode.

8085 µP	8086
→ It is 40 pin IC	→ It is 40 Pin IC
→ Based on NMOS tech.	→ Based on HMOS tech.
→ V_{cc} = +5V	→ V_{cc} = 5V
→ operating freq = 3MHz.	→ operating freq = 5MHz.
→ It is 8 bit µP.	→ It is 16 bit µP.
→ Add. Pin = 16, Data Pin = 8	→ Add. Pin = 20, Data Pin = 16

→ Max. memory that can interface = 64 K Byte	→ Max. memory that can interface = 1M Byte.
→ Flag Reg of 8 bit	→ Flag Reg 16 bit
→ No. of flags = 5	→ No. of flags = 9

(83)

Internal Arch. =>

→ Internal Arch. of 8086 divide in two parts.

(i) Bus interface unit (BIU).

(ii) Execution unit.

(i) BUS interface unit (BIU).

→ All type of data & add. transfer over buses managed by this part.

→ In this part there is a Queue of six locatic that is used for storage of instruction the fetch during the execution of current inst.

→ Queue works on the principle of FIFO.

(ii) EU (Execution unit)

→ All type of data execution occur in this part.

→ ALU of 8086 present in execution part.

Memory Segmentation

Note :- 8086 µP, 16 bit data can be read from memory (continuous location) in one r/c cycle, if 1st Byte at even addres

8086 requires 2 m/c cycle 16 bit data from memory if 1st Byte at odd address.

→ But in the case of 8088 it requires two m/c cycle to read 16 bit data from two continuous memory location (only 8 bit in one m/c cycle). but process 16 bit in one M-C, so it is externally 8 bit internally 16 bit μP.

Memory segmentation

→ 1 M Byte memory divide in four segment of size 64 kByte in continuous memory location & each segment used for storage of definite type of information.

- Code segment ≡ 64 k Byte ≡ Instructions feed in this seg
- Stack segment ≡ 64 k Byte ≡ Temporary information during the execution of main program.
- Data segment ≡ 64 kByte } data storage.
- Extra segment ≡ 64 k Byte }

$\frac{1 \text{ M Byte}}{64 \text{ k Byte}} = 16$ → But we use only 4.

→ segment can be defined any where in 1MByte memory.

→ Bottom add. of each seg select in such a way that lower 4 bit should be zero.

→ Upper 16 bit of bottom add. store in a specific type of reg. dedicated to segment.

Code segment ≡ CS (16 bit).
Stack segment ≡ SS (16 bit).
Data segment ≡ DS (" ").
Extra segment ≡ ES (" ").

→ Regarding each segment there is another 16 bit reg. that hold the 16 bit add. in rang of 64 KByte (0000 H to FFFF H).

Code segment ≡ Instruction Pointer (IP)
Stack segment ≡ Stack pointer (SP)
Data segment ≡ Source index (SI)
Extra segment ≡ Destination index (DI)
} 16 bit

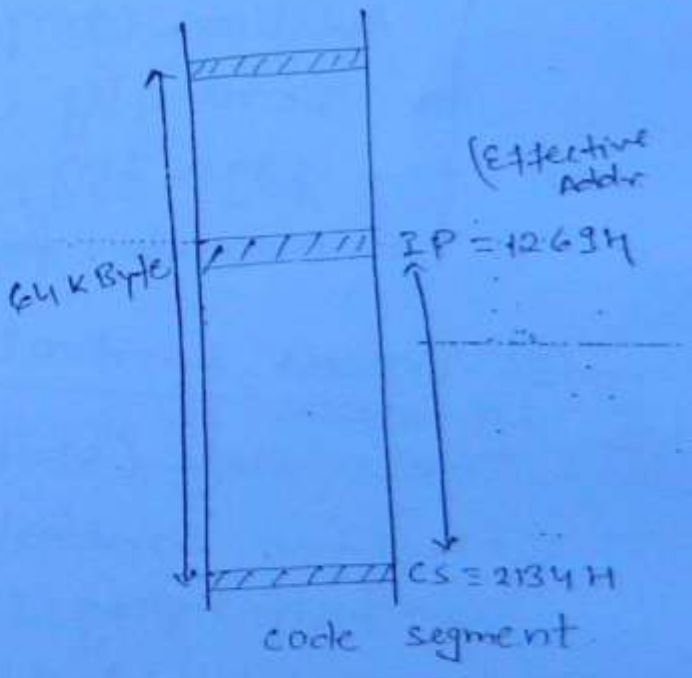
* Physical Address

Ext Actual memory Add
Memory location Add
}

→ Instruction Pointer (I.P.) is similar as program counter (P.C.) of 8085 MP.

CS : IP
offset add.

CS = 2134 → Hard wired zero.
IP = 1269
Physical Add. = 225A9 H



Flag Reg:

→ Size of flag reg is 16 bit.

→ All five flags of 8085 common in 8086 µp.

→ Total no. of flags in 8086 is nine.

→ Nine flags can be divicle in two categoryies

→ ① Conditional flag :- Status of these flags affect accoroling to condition generate in A & L operations.

No. of conditional flag = 6.

- Carry flag (CY)
- Parity flag (P)
- Auxiliary Carry flag (AC)
- Zero flag (Z)
- Sign flag (S)
- over flow flag (O).

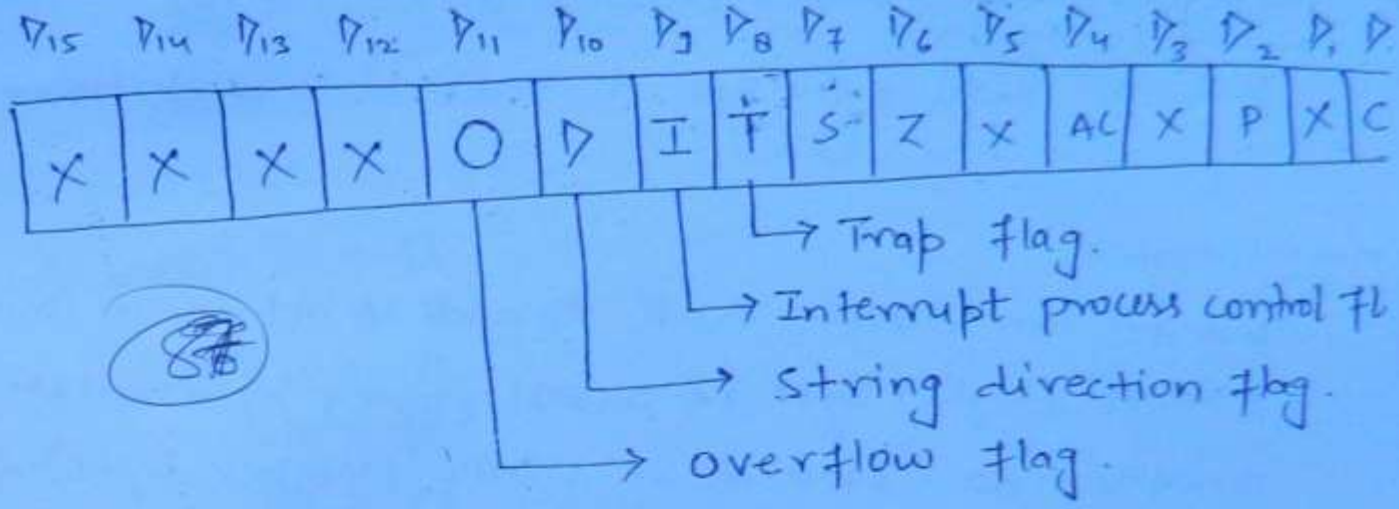
→ ② Process Control flag:

These flags (status of these flags) used for machine control / or process controlling.

→ Trap flag (T)

→ Interrupt flag (I)

→ String direction flag (D).



Addressing mode

Form of effective add. present in the instr. is known as add. mode.

① Immediate Add. Mode

If data itself given in the inst. then immediate add. mode (in the operand.).

② Direct Addl. Mode

If effective add itself part of inst. then it is known as direct add. mode.

③ Indirect Reg. Addl. mode

If EA is given in the form of content of reg

Imp.

④ Register relative Addl. Mode

Relative ≡ 8 or 16 bit displacement no.

EA = [BX] + 8 or 16 bit displacement no.

Note: BH BL ≡ BX.

AH AL ≡ AX.

⑤ Base Index add. mode

EA = [Bx] + [SI] or [DI].

```
LXI B, 2001 H
LXI D, 3001 H.
MVI H, 0A H
Loop. : LDAX B.
        STA X D.
        INX B
        INX D.
        DCR H
        JNZ loop.
        HLT.
```

Q:

```
        ORG   7000H
BEGIN:  LXI H, 7000H     [H]=70, [L] = 00H
(7000H) MOV A, L         [A] = 00H
        ADD H            [A] = 0000 0000
        JP END                 0111 0000
        RST 0            [A]  0111 0000
END:    PC HL                    S=0.
        HLT
```

PC = 7000H

Infinite times.

W.B. Page 100.

$(XX)_H$ = (H) decimal

Q: 20.

```
        MVI B XXH     = FR = 7T
L2      MVI C, FFH    = FR = 7T <---
L1      DCR C         = F  = 4T <---
        JNZ L1        = FRR/FR = 10T/7T
        DCR B         = F = 4T
        JNZ L2        = FRR/FR = 10T/7T <
        HLT           = 4T/5T.
```

Inner loop completel exe = 3567 T

100×10^3 = 7T + (N-1) [7T + 3567 T + 4T + 10T

T = D.C. per + 1 [7T + 3567 T + 4T + 7T] + 4